НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.94:544.182.37

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету	Завідувач кафедри комп'ютерних наук
інформаційних технологій	
Глазунова О.Г., д.п.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ **2021 р.**	**«30» листопада 2021 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему «Інтелектуальна система швидкого шифрування дискретних даних»

Спеціальність 122 «Комп'ютерні науки»

Освітня програма «Інформаційні управляючі системи та технології»

Орієнтація освітньої програми освітньо-професійна

Гарант освітньої програми «Інформаційні управляючі системи та технології»

д.т.н, доцент _____ Бондаренко В.Є.

Керівник магістерської кваліфікаційної роботи

к.ф.м.н., с.н.с., _____ Лялецький О.В.

Виконав _____ Сохацький Б.Д.

КИЇВ-2021

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

Голуб Б.Л., к.т.н., доцент

"29" жовтня 2020 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Сохацькому Богдану Дмитровичу

Спеціальність 122 «Комп'ютерні науки»

Освітня програма «Інформаційні управляючі системи та технології»

Орієнтація освітньої програми освітньо-професійна

Тема магістерської кваліфікаційної роботи «Інтелектуальна система швидкого шифрування дискретних даних»

затверджена наказом ректора НУБіП України від "29" жовтня 2020 р. № 1634 «С»

Термін подання завершеної роботи на кафедру «30» листопада 2021 р.

Вихідні дані до магістерської кваліфікаційної роботи

1) Дані проіснуючі алгоритми та системи шифрування.
2) Якісні та кількісні характеристики сучасних алгоритмів шифрування дискретних даних.

Перелік питань, що підлягають дослідженню:

№ з/п	Питання, що підлягає дослідженню	Строк виконання	Примітка
1.	Аналіз предметної області.	21.09.2020 – 20.10.2020	
2.	Дослідженя існуючих алгоритмів шифрування дискретних даних	25.10.2020 – 21.01.2021	
3.	Розробка та програмна реалізація алгоритму швидкого шифрування дскретних даних	01.02.2021 – 01.06.2021	
4.	Дослідження отриманих результатів	02.06.2021 – 01.09.2021	
5	Попередній захист	30.11.2021	
6.	Захист	14.12.2021	

Дата видачі завдання "29" жовтня 2021 р.

Керівник магістерської кваліфікаційної роботи _____ Лялецький О.В.
(підпис)

Завдання прийняв до виконання _____ Сохацький Б.Д.
(підпис)

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Інтенсивний розвиток інформаційних технологій неминуче призводить до задачі забезпечення конфіденційності і цілісності інформації.

Технології віртуальної інфраструктури неминуче впроваджуються в сучасне життя. Експлуатація інформаційних систем в контексті обробки даних стає звичним явищем, зручним, звичним і економічно виправданим. Комплексне забезпечення інформаційної безпеки являє собою безперервний процес, який постійно вимагає модифікації і, як результат, постійного ускладнення IT-систем.

Інфраструктура IT-систем має на увазі впровадження технічних або програмних компонентів, наявність яких призводить до збільшення можливостей несанкціонованого доступу до оброблюваної інформації.

Інформація стає стратегічним ресурсом держави і бізнесу всіх рівнів, які зацікавлені в її збереженні. Крім природних ризиків втрати інформації (відмова техніки, стихійні лиха і т. д.), присутнє також прагнення кримінальних структур здійснити незаконне викрадення або модернізацію інформації.

У світлі сказаного проблема захисту інформації є надзвичайно актуальною на сьогоднішній день Питанням захисту інформації протягом всієї історії приділялася пильна увага для нормалізації функціонування життя суспільства.

При побудові систем захисту використовуються програмні рішення в області інформаційної безпеки, які дозволяють налаштувати політику безпеки на підприємстві, централізовано керувати процесами захистів, інтегрувати різні механізми в єдину систему і виділяти різні ролі для адміністрування системи. Існує безліч способів здійснити атаку на корпоративну мережу підприємства.

Криптографічні методи і засоби захисту інформації, а також їх математичні основи є фундаментальними дослідженнями, що зв'язують воєдино області математики, інформатики та фізики. Основною задачею шифрування є забезпечення конфіденційності інформації, що передається.

Основна мета криптографії полягає не тільки в забезпеченні конфіденційності, а й для вирішення інших завдань, таких як: збереження цілісності даних, автентифікації, фіксації авторства.

В даний час тривають безперервні дослідження нових криптографічних алгоритмів. Проте, важко підібрати конкретний алгоритм, так як вже відомо, що вони повинні враховувати безліч факторів: безпека, особливості алгоритму, трудомісткість і ресурсомісткість.

Саме тому дослідження методів шифрування дискретних даних та розробка такого методу, який буде задовольняти сучасним вимогам до захисту інформації, є актуальним напрямком досліджень.

Метою написання магістерської дисертації є розробка інтелектуальної системи швидкого шифрування дискретних даних.

При написанні роботи були поставлені наступні задачі:

- виконати огляд літератури з теми дослідження. Навести існуючі алгоритми шифрування дискретних даних;
- навести моделювання інтелектуальної системи шифрування дискретних даних;
- представити результати дослідження інтелектуальної системи шифрування дискретних даних.

Об’єктом дослідження магістерської дисертації є процес шифрування дискретних даних.

Предметом дослідження магістерської дисертації є інтелектуальна система швидкого шифрування дискретних даних.

При написанні магістерської дисертації були використані наступні методи дослідження:

- метод аналізу;

- метод статистики,
- метод проектування.

Наукова новизна отриманих результатів полягає в тому, що розроблено інтелектуальну система швидкого шифрування дискретних даних.

Магістерська дисертація складається з трьох розділів. Перший розділ присвячений огляду сучасних алгоритмів шифрування дискретних даних. У другому розділі роботи наведено розробку інтелектуальної системи швидкого шифрування дискретних даних. Третій розділ показує практичні результати дослідження інтелектуальної системи шифрування дискретних даних.

Апробація результатів дослідження. За матеріалами магістерської дисертації були сформовані тези з тем «Про надійність та ефективність криптографічних алгоритмів» та «Проблема вибору системи шифрування».

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості шифрування дискретних даних

В межах даного розділу роботи розглянемо поняття дискретних даних та особливості їх шифрування.

Дискретні дані - дані, число яких скінченне або нескінченне, але може бути підраховано за допомогою натуральних чисел від одного до нескінченності. Дискретними є всі дані рядкового чи логічного типу. Дискретними можуть бути і числові дані, наприклад, " Код товару ", який приймає значення цілого типу [1].

Дискретні дані задаються зазвичай у формі цілих чисел. Ці значення повинні відповідати певним класифікаціям і не можуть бути розбиті на дрібніші частини.

Деякі приклади дискретних даних можуть включати:

- Кількість співробітників у відділі;
- Кількість нових клієнтів;
- Кількість товарів, що знаходяться в даний час на складі.

У всіх цих прикладах описане окреме значення, яке можна підрахувати та присвоїти фіксоване числове значення [2].

Дискретні та неперервні дані зазвичай плутають один з одним через їх схожість з числовими типами даних. Однак основна відмінність між дискретними та неперервними даними полягає в тому, що дискретні дані - це кінцеве значення, яке можна підрахувати, тоді як неперервні дані мають нескінченну кількість можливих значень, які можна виміряти.

Шифрування дискретних даних перетворює дані з формату відкритого тексту, що читається, в нечитаний закодований формат: зашифрований текст. Користувачі та процеси можуть читати та обробляти зашифровані дані тільки після їх розшифрування. Ключ дешифрування є секретним, тому він має бути захищеним від несанкціонованого доступу.

Шифрування дискретних даних - це процес перетворення даних на непридатну для використання форму, який сам по собі не зупиняє злом або крадіжку даних. Натомість він запобігає використанню вкраденого контенту, оскільки хакер або злодій не можуть бачити його у форматі відкритого тексту.

Актуальність шифрування дискретних даних пояснюється тим, що величезні обсяги конфіденційної інформації управляються та зберігаються в Інтернеті або на підключених серверах. Це пов'язано з тим, що практично неможливо вести життя день у день без передачі та зберігання конфіденційних даних у мережевих комп'ютерних системах різних організацій.

Алгоритми шифрування дискретних даних шифрують відкритий текст, тому тільки людина з ключем дешифрування може його прочитати. Цей процес забезпечує безпеку даних для особистої інформації, яку користувачі отримують, надсилають та зберігають на мобільних пристроях, у тому числі підключених до IoT [3].

Технологія шифрування дискретних даних захищає як передані дані, так і збережені цифрові дані в комп'ютерних системах та у хмарі. У міру того, як Інтернет змінив обчислення та системи стали онлайн, сучасні алгоритми шифрування (шифри) замінили застарілі стандарти шифрування даних для захисту IT-комунікацій та систем.

Ці алгоритми шифрування дискретних даних забезпечують конфіденційність та підтримують основні ініціативи щодо забезпечення безпеки, включаючи цілісність, автентифікацію. Алгоритми спочатку аутентифікують будь-яке повідомлення, щоб перевірити його походження, а потім перевіряють його цілісність, щоб переконатися, що вміст залишився незмінним під час передачі. Зрештою, ініціатива запобігання відмови не дозволяє відправникам заперечувати законну діяльність.

Існує два основних типи шифрування дискретних даних: симетричне шифрування та асиметричне шифрування. При симетричному шифруванні один пароль шифрує та дешифрує дані. Асиметричне шифрування, що іноді називається шифруванням з відкритим ключем або криптографією з відкритим

ключем, використовує два ключі для шифрування та дешифрування. Загальний відкритий ключ шифрує дані. Закритий не загальний ключ, який повинен залишатися захищеним, розшифровує дані.

Шифрування із симетричним ключем відбувається швидше, ніж асиметричне шифрування, але перед тим, як розшифрувати, воно вимагає, щоб відправник обмінявся ключем шифрування з отримувачем. Це, у свою чергу, призвело до появи величезної кількості ключів для безпечного управління організаціями - проблеми, що зростає. Тому багато служб шифрування даних адаптувалися до використання асиметричних алгоритмів.

Крім симетричної та асиметричних алгоритмів шифрування дискретних даних, сьогодні на практиці існує кілька методів шифрування та обробки захищених даних. Кожен стандарт шифрування даних був розроблений для задоволення різноманітних вимог безпеки. Найбільш поширені приклади методів шифрування даних будуть розглянуті у наступному розділі роботи.

Основні переваги шифрування даних полягають у наступному:

- можна використовувати інструменти та технології шифрування даних на кількох пристроях. Це дає можливість працювати з дому та розширює можливості електронної комерції;

- шифрування даних забезпечує безпечніший зв'язок. Шифрування даних підвищує онлайн-безпеку, гарантуючи, що всі повідомлення, що передаються, не можуть бути прочитані неавторизованими користувачами, оскільки дані були зашифровані;

- злом – серйозний ризик. Кіберзлочинність та великомасштабні витоки даних є постійним ризиком для будь-якої організації, і навіть найнадійніші засоби захисту можуть бути зламані. Це означає, що необхідно виходити з того, що конфіденційні дані можуть бути втрачені, перехоплені або вкрадені - і вони повинні бути зашифровані;

- шифрування дискретних даних підтримує цілісність даних.

Шифрування дискретних даних може захищати інтелектуальну власність. Системи управління цифровими правами шифрують неактивні дані.

Шифрування дискретних даних забезпечує надійну безпеку як для людей, які працюють із конфіденційними даними, так і для тих, хто довіряє свої дані іншим.

Шифрування дискретних даних для підприємств гарантує, що підприємства дотримуються застосовних нормативних стандартів. Це також допомагає захистити цінні дані своїх клієнтів. Це деякі з причин, з яких шифрування даних важливе [3].

Рішення для шифрування дискретних даних, такі як програмне забезпечення для шифрування даних та хмарне шифрування даних, часто поділяються на категорії.

Оскільки співробітники, які працюють з дискретними даними, все частіше використовують знімні носії, зовнішні пристрої та веб-додатки в рамках поточних операцій, рішення для шифрування даних повинні справлятися з зростаючими організаційними проблемами запобігання втраті даних та захисту самих даних. Як тільки співробітники переносять дані на мобільні пристрої або хмару, компанії можуть втратити контроль над ними.

З цих причин шифрування дискретних даних є важливим компонентом кращих рішень щодо запобігання втраті даних. Вони призначені для запобігання впровадженню хакерами шкідливих програм та крадіжки даних з хмарних та веб-додатків, а також зовнішніх та знімних пристроїв. Тим не менш, частина захисту даних полягає в тому, щоб вони не могли бути використані хакерами, якщо вони потраплять у чужі руки.

1.2 Аналіз наявних рішень шифрування дискретних даних

В межах даного розділу розглянемо існуючі та найбільш застосовувані алгоритми шифрування дискретних даних.

Як вже зазначалося, сучасні схеми шифрування використовують концепції відкритого ключа та симетричного ключа. Основні відмінності

симетричних та асиметричних алгоритмів, які можна застосовувати для шифрування дискретних даних, представлені у вигляді таблиці 1.1.

Таблиця 1.1 – Основні відмінності симетричних та асиметричних алгоритмів

Критерій порівняння	Симетричне шифрування	Асиметричне шифрування
1. Кількість ключів	Використовується лише один ключ (симетричний ключ), і той самий ключ використовується для шифрування та дешифрування повідомлення.	Два різні криптографічні ключі (асиметричні ключі), які називаються відкритим та приватним ключами, використовуються для шифрування та дешифрування.
2. Складність та швидкість виконання	Це проста техніка, і завдяки цьому процес шифрування може бути здійснений швидко.	Це набагато складніший процес, ніж симетричне шифрування, і процес відбувається повільніше.
3. Довжина ключів	Довжина використовуваних ключів, як правило, становить 128 або 256 біт, залежно від вимог безпеки.	Довжина ключів набагато більша, наприклад, рекомендований розмір ключа RSA становить 2048 біт або вище.
4. Використання	В основному використовується, коли потрібно передати великі об’єми даних.	Використовується в невеликих транзакціях, насамперед для автентифікації та

		встановлення безпечного каналу зв'язку.

Продовження табл.1.1

5. Безпека	Спільний секретний ключ. Процес є менш захищеним у порівнянні з асиметричним шифруванням.	Приватний ключ не використовується спільно, і загальний процес є більш безпечним у порівнянні з симетричним шифруванням.
6. Приклади алгоритмів	CAST-128, BLOWFISH, IDEA, RC2, RC4, RC5, RC6, AES, DES, 3DES тощо.	RSA, Diffie-Hellman, ECC тощо.

Розглянемо ці алгоритми.

Алгоритм CAST-128. CAST-128, також відомий як CAST5, є блоковим шифром, що використовується в ряді продуктів. CAST-128 був створений в 1996 році Карлайлом Адамсом і Стаффордом Таваресом.

CAST-128 - це мережа Фейстеля з 12 або 16 циклами з розміром блоку 64 біт та розміром ключа від 40 до 128 біт (але тільки з кроком 8 біт). Якщо розмір ключа перевищує 80 біт, використовуються повні 16 раундів. Компоненти включають великі 8 x 32-бітові S-блоки, засновані на функціях вигину, які залежні від ключа обертання, модульному складанні і відніманні і операціях XOR. Є три типи раундової функції, але вони схожі за структурою і відрізняються тільки вибором точної операції (складання, віднімання або XOR) у різних точках.

CAST-256, також відомий як CAST6, був отриманий із CAST-128 та опублікований у червні 1998 року. Він був представлений як кандидат на Advanced Encryption Standard (AES). CAST-256 використовує ті ж елементи, що і CAST-128, включаючи S-блоки, але адаптовані для розміру блоку 128 біт - удвічі більше, ніж у 64-бітного попередника. Допустимі розміри ключа: 128, 160, 192, 224 або 256 біт. CAST-256 складається з 48 раундів.

Блок-схема алгоритму представлена на рис.1.1.

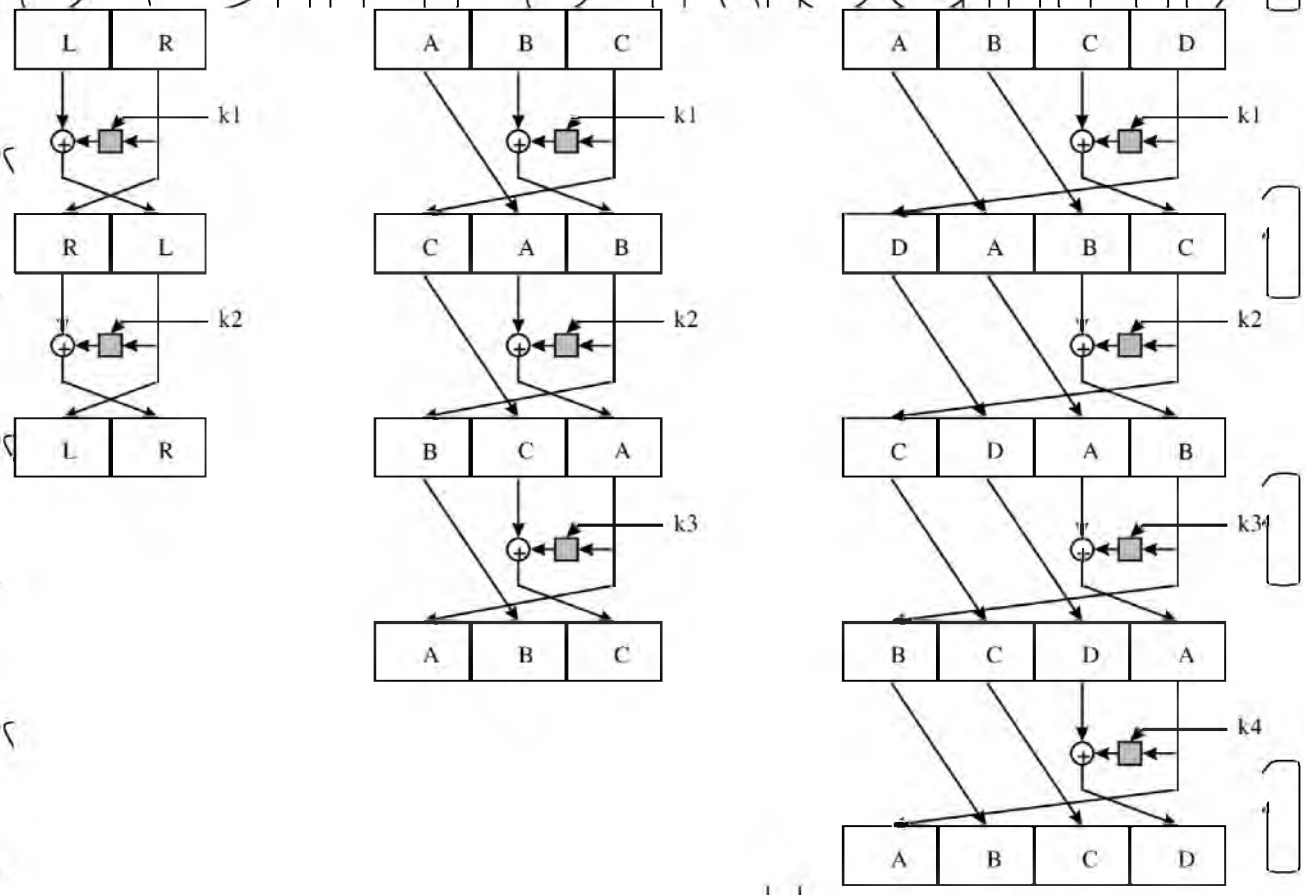

Рис.1.1. Блок-схема алгоритму

Як для шифрування (encrypt()), так і для дешифрування (decrypt()) алгоритм CAST-128 приймає на вхід ключ key і msg, що кодується.

Принцип роботи функцій encrypt() і decrypt() дуже схожий, але є й відмінність: для розшифровки потрібно робити те саме, що й при шифруванні, але у зворотному порядку. Для цього при програмній реалізації необхідно створити допоміжну функцію run(), якою можна передати додатковий прапор reverse (false для шифрування та true для дешифрування).

Передбачається, що шифрування та дешифрування здійснюється "на місці". Тобто, повідомлення msg кодуються без копіювання.

Blowfish. Blowfish – це симетричний блоковий шифр, який можна використовувати як заміну DES або IDEA. Він приймає ключ змінної довжини, від 32 до 448 біт, що робить його ідеальним як для домашнього, так експортного використання. Blowfish був розроблений Брюсом Шнайером у 1993 році як швидка безкоштовна альтернатива існуючим алгоритмам шифрування. З того часу він зазнав значного аналізу і поступово отримує визнання як надійний алгоритм шифрування. Blowfish не запатентований, не вимагає ліцензії та доступний безкоштовно для будь-якого використання.

При проектуванні Blowfish використовувалися такі критерії [4]:

- Швидкість. Blowfish шифрує дані на 32-бітових мікропроцесорах із швидкістю 26 тактів на байт.

- Компактність. Blowfish може займати менше, ніж 5 Кбайт пам'яті.

- Простота. Blowfish використовує лише прості операції: додавання, XOR та вибірка з таблиці за 32-бітовим операндом. Аналіз його схеми нескладний, це зменшує кількість помилок під час реалізації.

- Безпека. Довжина ключа Blowfish є змінною і може досягати 448 бітів.

Blowfish є 64-бітовим блоковим шифром з ключем змінної довжини. Алгоритм складається з двох частин: розгортання ключа та шифрування даних. Розгортання ключа перетворює ключ довжиною до 448 бітів на кілька масивів підключів, загальним обсягом 4168 байтів.

Алгоритм шифрування представлений на рис.1.2.

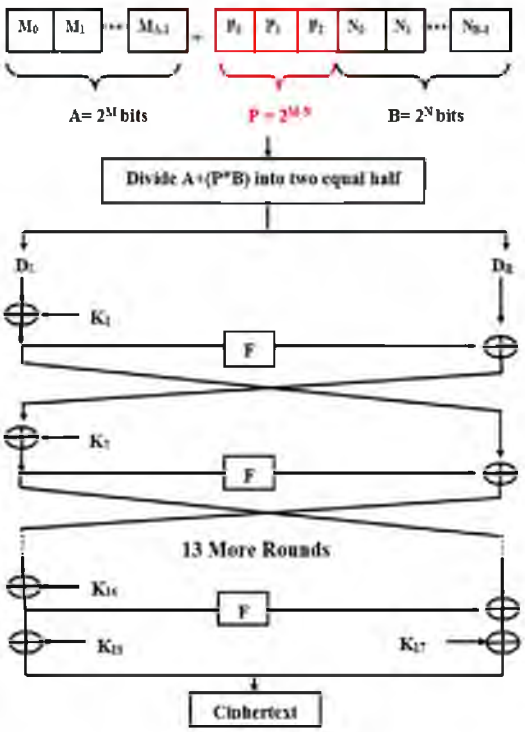

Рис.1.2. Блок-схема алгоритму шифрування Blowfish

Алгоритм шифрування даних (IDEA) – це колись пропрієтарний безкоштовний та відкритий блоковий шифр, який колись призначався для заміни стандарту шифрування даних (DES). Колись званий покращеним пропонованим стандартом шифрування (IPES) IDEA є незначною редакцією стандарту шифрування (PES).

Алгоритм шифрування представлений на рис.1.3.

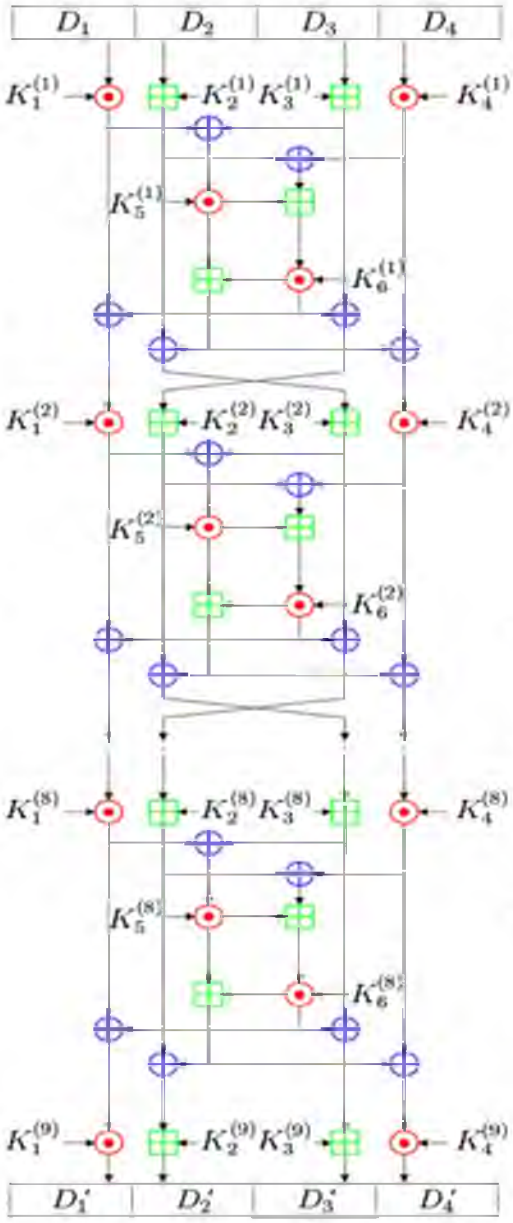

Рис. 1.3. Алгоритм шифрування IDEA

IDEA використовує аналогічні процеси для шифрування та дешифрування з деяким зворотним порядком раундових ключів. Алгоритм складається із серії з 8 циклів і працює з 64-бітними блоками з використанням 128-бітного ключа. IDEA «страждала» від слабких ключів до тих пір, поки не було переглянуто її розклад ключів, і в майбутньому може знадобитися подальший перегляд.

Характеристики:

- використовує відкритий текст фіксованої довжини з 16 біт і шифрує їх 4 блоками по 4 біти кожен для створення 16-бітного зашифрованого тексту;
- довжина використовуваного ключа становить 32 біти.

Ключ також поділено на 8 блоків по 4 біти в кожному.

Цей алгоритм включає серію з 4 ідентичних повних раундів та 1 півкола. Кожен повний раунд включає серію з 14 кроків.

Після 4 повних раундів фінальне «півколо» складається лише з перших 4 з 14 кроків, які раніше використовувалися в повних раундах. Для виконання цих циклів кожне двійкове уявлення має бути перетворене на його еквівалентне десяткове уявлення, виконати операцію, і отриманий результат повинен бути перетворений назад у двійкове уявлення для остаточного результату цього конкретного кроку.

Розклад ключів: 6 підключів по 4 біти із 8 підключів використовуються в кожному повному раунді, а 4 - у півкрузі. Отже, для 4,5 раундів потрібно 28 підключів. Цей ключ, "K", безпосередньо дає перші 8 підключів. Шляхом повороту основного ключа вліво на 6 бітів між кожною групою з 8 створюються додаткові групи з 8 підключів, що має на увазі менше одного повороту за раунд для ключа (3 повороти).

16-бітовий відкритий текст може бути представлений як X1 | X2 | X3 | X4 кожен розміром 4 біта. 32-бітний ключ розбитий на 8 підключів, позначених як K1 | K2 || K3 || K4 || K5 || K6 || K7 || K8, знову розміром 4 біти кожен. У кожному раунді з 14 кроків використовуються три операції алгебри: додавання по модулю (2 ^ 4), множення по модулю (2 ^ 4) +1 і побітове виключення.

Далі розглянемо алгоритми RC2, RC4, RC5, RC6.

Алгоритм RC2 шифрує дані блоками по 64 біти з використанням ключів змінного розміру: від 8 до 1024 бітів виключно; рекомендованим розміром ключа є 64 біти.

Алгоритм є мережею Фейстеля, у ньому виконується 18 раундів перетворень. Причому раунди алгоритму діляться на 2 типи: раунди, що змішують (mix) і раунди, що об'єднують (mesh). Загальна структура алгоритму така:

- Виконується 5 раундів, що змішують.
- Виконується 1 об'єднуючий раунд.
- Виконуються 6 раундів, що змішують.
- Виконується 1 об'єднуючий раунд.
- Виконується 5 раундів, що змішують.

Структура змішуючого раунду наведена на рис.1.4. Передбачається, що блок даних, що шифрується, розділений на 4 16-бітних слова , над якими змішуючий раунд в циклі по і від 0 до 3 виконує наступні операції:

$$R_i = R_i + K_j + (R_{i-1 \bmod 4} \& R_{i-2 \bmod 4}) + (\sim R_{i-1 \bmod 4} \& R_{i-3 \bmod 4}) \bmod 2^{16},$$
$$j = j + 1,$$
$$R_i = R_i \lll S_i, \quad (1.1)$$

де K_j – фрагмент розширеного ключа, який визначається глобальною змінною j; дана змінна спочатку дорівнює нулю і збільшується на 1 (як показано вище) в кожному раунді, що змішує; процедура розширення ключа докладно описана далі,

$\&$ - Побітова логічна операція «і»,

$\sim x$ - Побітовий комплемент до x,

«- циклічне зрушення вліво на число бітів, що визначається значенням S_i (табл. 1).

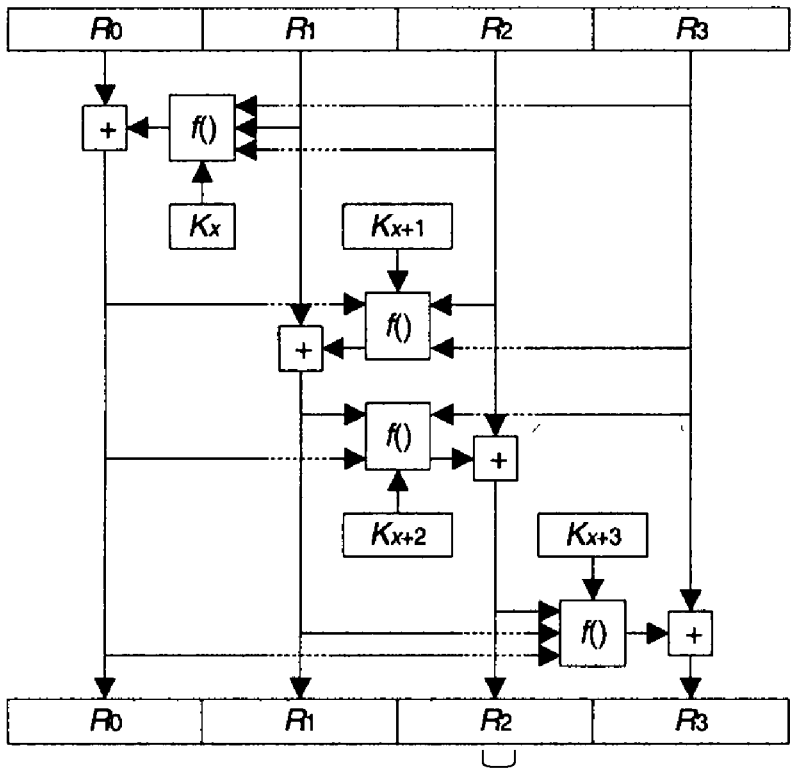

Рис.1.4. Змішуючий раунд алгоритму RC2

Таблиця 1.2 – Значення S_i

i	0	1	2	3
S_i	1	2	3	5

Таким чином, у кожній i-й ітерації змішувального раунду виконується описане вище перетворення f() (рис.1.5), яке модифікує R_i на основі поточних значень трьох інших слів блоку, що шифрується, і фрагмента розширеного ключа.

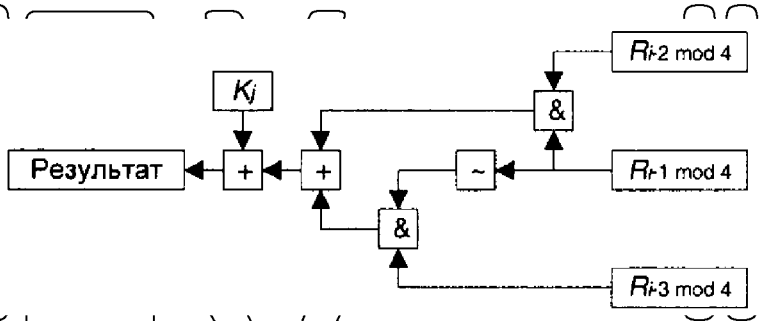

Рис.1.5. Функція f() алгоритму RC2

Аналогічно змішуючому раунду, в об'єднуючому раунді виконується цикл і від 0 до 3; у кожній ітерації циклу виконується наступна операція:

$$R_i \leftarrow R_i + K_n \bmod 2^{16}, \quad (1.2)$$

де $n = R_{i-1 \bmod 4} \& 63$

Таким чином, ітерація об'єднуючого раунду - це накладання операцією додавання по модулю 2^{16} фрагмента розширеного ключа, індекс якого визначається 6 молодшими бітами поточного значення $R_{i-1 \bmod 4}$.

RC4 є потоковим шифром зі змінною довжиною ключа, розроблений в 1987 р. Роном Рівестом для компанії RSA Data Security, Inc.

Розглянемо опис принципу шифрування. Ключова послідовність залежить від вихідного тексту. Структура алгоритму включає блок заміни розмірністю 8x8: $S_0 \dots S_{255}$. Блок заміни є залежним від ключа змінної довжини перестановки чисел 0,…, 255. Є два лічильники і і j, спочатку рівні 0. Для генерування псевдовипадкового байта виконуються такі дії:

$$i = (i + 1) \bmod 256, \quad (1.3)$$

$$j = (j + S_i) \bmod 256,$$

переставити S_i таS_j

$$t = (S_i + S_j) \bmod 256, \quad (1.4)$$

$$k = S_t. \quad (1.5)$$

Потім байт k складається по модулю 2 з байтом вихідного тексту отримання шифрованого.

Ініціалізація блоку заміни також проста. Спочатку він заповнюється лінійно: $S_0 = 0, S_1 = 1, \dots S_{255} = 255$. Потім заповнюється ще один 256-байтний масив ключем, при цьому ключ може повторюватися необхідне число разів для заповнення всього масиву: $k_0 \dots k_{255}$. Лічильник j встановлюється в 0. Після чого виконуються такі дії:

for i=0 to 255

$$j = (j + k_i + S_i) \bmod 256$$

переставити S_i та S_j.

Можливе узагальнення алгоритму на більшу довжину і розмір блоку заміни. Так, можна побудувати шифр із блоком заміни розмірністю 16x16 (потрібно 128 Кбайт пам'яті) та довжиною 16 біт. Етап ініціалізації буде значно повільнішим, необхідний цикл до 65535, якщо ми прагнемо в точності слідувати конструкції, але алгоритм, що вийшов в результаті, буде більш швидким.

Алгоритм RC5 розроблений найвідомішим криптологом Рональдом Рівестом – одним із розробників асиметричної системи RSA. Аналогічно попереднім алгоритмам шифрування Рона Рівеста RC2 і RC4 (є потоковим шифром), алгоритм RC5 отримав дуже широке поширення.

На перетвореннях, що використовуються в RC5, засновано подальшу розробку компанії RSA – алгоритм RC6, який став фіналістом конкурсу AES щодо вибору нового стандарту шифрування США [5].

Частина основних параметрів алгоритму RC5 є змінними. Крім секретного ключа, параметрами алгоритму є:

- розмір слова w (у бітах); RC5 шифрує блоками два набори дискретних даних; допустимими значеннями w є 16, 32 або 64, причому 32 є рекомендованим;

- кількість раундів алгоритму R – як значення припустимо будь-яке ціле число від 0 до 255 включно;

- розмір секретного ключа в байтах b – будь-яке значення від 0 до 255 включно.

Найчастіше для уточнення параметрів алгоритму, у його конкретній реалізації, застосовується позначення RC5-w/R/b; наприклад, RC5-32/12/16 позначає RC5 з 64-бітним блоком, 12 раундами та 128-бітним (16-байтним) ключем.

Змінні параметри розширюють сферу використання алгоритму, а також дозволяють сильно скоротити витрати при необхідності переходу на сильніший варіант алгоритму, програмної або апаратної реалізації RC5, що

підтримує змінні параметри, легко було б замінити ключ довшим, таким чином усунувши проблему.

Передбачена в алгоритмі і проблема сумісності реалізацій RC5 з різними параметрами – кожне зашифроване повідомлення рекомендується передувати заголовком, що містить список значень основних параметрів алгоритму – передбачається, що в цьому випадку для розшифровування повідомлення слід встановити параметри із заголовка, після чого (за наявності коректного ключа) повідомлення легко буде розшифровано.

Структура алгоритму RC5 наведена на рис.1.6.

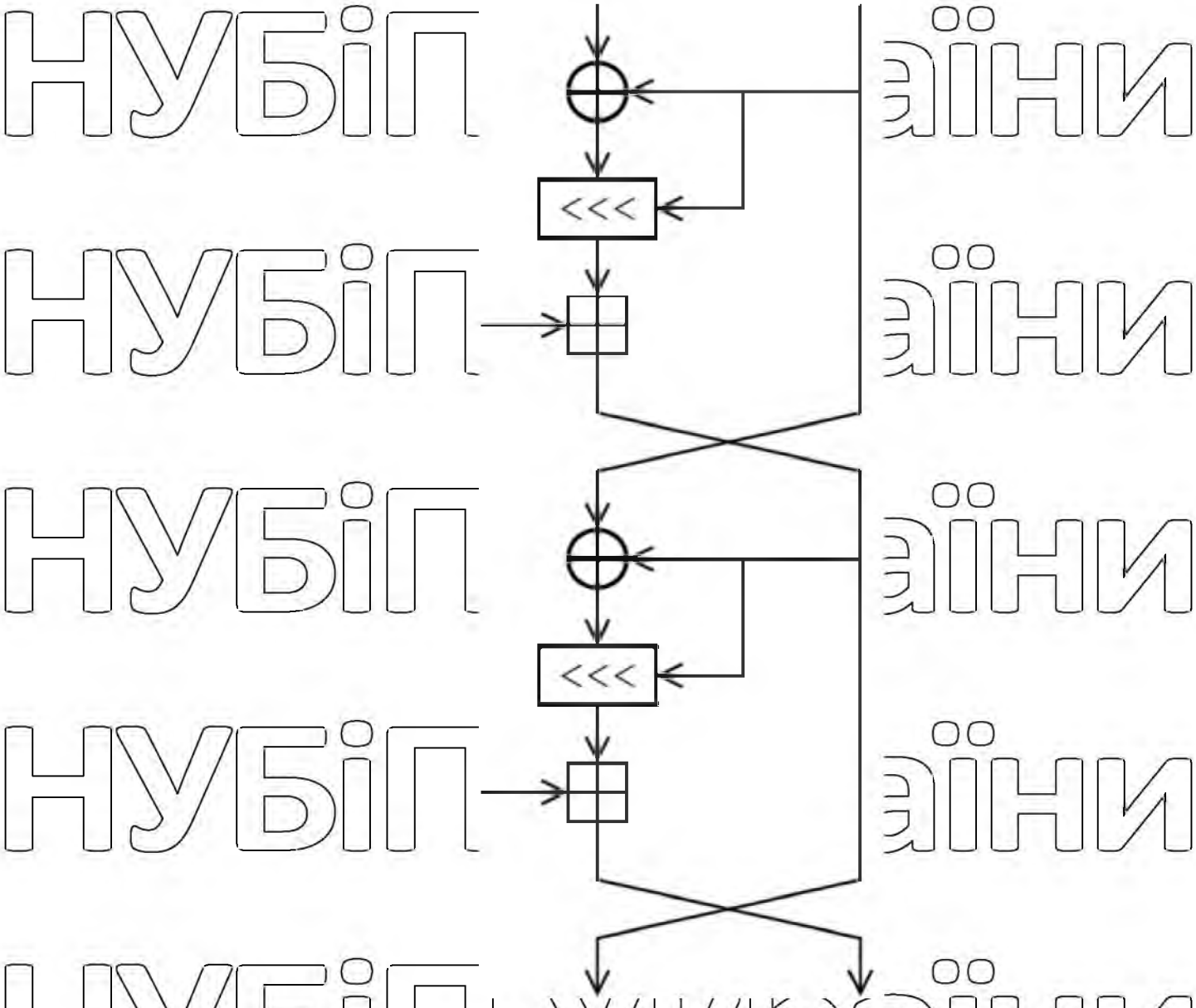

Рис.1.6. Структура алгоритму RC5

Алгоритм є мережею Фейстеля, у кожному раунді якої виконуються такі операції:

$$A = \big((A \oplus B) \ll B\big) + K_{2*r} \; mod \; 2^w, \tag{1.6}$$

$$B = \big((A \oplus B) \ll A\big) + K_{2*r+1} \; mod \; 2^w, \tag{1.7}$$

де r – номер поточного раунду, починаючи з 1,

K_n - фрагмент розширеного ключа,

$\ll n$ – Операція циклічного зсуву на x бітів вліво, де x - значення молодших $log_2 w$ бітів n.

Перед першим раундом виконуються операції накладання двох перших фрагментів розширеного ключа на дані, що шифруються:

$$A = A + K_0 \; mod \; 2^w, \tag{1.8}$$

$$B = B + K_1 mod \; 2^w. \tag{1.9}$$

Під словом «раунд» в описі алгоритму розуміються перетворення, що відповідають двом раундам звичайних алгоритмів, структура яких мережа Фейстеля (рис.1.7).

Тобто раунд алгоритму RC5 обробляє блок повністю, тоді як типовий раунд мережі Фейстеля обробляє лише один субблок – зазвичай половину блоку.

Алгоритм простий – у ньому використовуються тільки операції додавання по модулю 2 і 2^w по модулю, а також зрушення на змінну кількість бітів. Зрушення на змінну кількість бітів є дуже просто реалізованою операцією, яка, проте, істотно ускладнює диференціальний та лінійний криптоаналіз алгоритму. Простота алгоритму може розглядатися як його важлива перевага – простий алгоритм легше реалізувати та легше аналізувати щодо можливих уразливостей [6].

Розшифровування виконується застосуванням зворотних операцій на зворотній послідовності, тобто, у кожному раунді r (із зворотною послідовністю раундів) виконуються такі операції:

$$B = \big((B - K_{2*r+1} \; mod \; 2^w) \gg A\big) \oplus A, \tag{1.10}$$

$$A = ((A + K_{2*r} \bmod 2^w) \gg B) \oplus B, \quad (1.11)$$

де »n - аналогічна описаній вище («n) операція побітового циклічного зсуву вправо.

Відповідно, після R раундів виконуються такі операції:

$$B = B - K_1 mod\ 2^w, \quad (1.12)$$

$$A = A - K_0\ mod\ 2^w. \quad (1.13)$$

Алгоритм RC5 та деякі його варіанти є запатентованими.

Структура алгоритму RC5, незважаючи на свою простоту, була багатьом криптологам як поле для можливих удосконалень. Відповідно, з'явилося безліч відомих варіантів алгоритму RC5, у яких перетворення в «півраундах» класичного RC5 дещо змінені:

1. Алгоритм RC5XOR, у якому додавання з ключем раунду по модулю 2^w замінено операцією XOR (рис.1.7):

$$A = ((A \oplus B) \ll B) \oplus K_{2*r}. \quad (1.14)$$

Даний алгоритм виявився менш стійким, ніж RC5, як до лінійного, так і до диференціального криптоаналізу.

2. Алгоритм RC5P, в якому додавання лівого та правого оброблених субблоків операцією XOR замінено додаванням по модулю 2^w (рис. 1.7):

$$A = ((A + B \bmod 2^w) \ll B) + K_{2*r} \bmod 2^w. \quad (1.15)$$

Алгоритм виявився так само стійкий, як і RC5, проти лінійного криптоаналізу, але значно слабше проти диференціального.

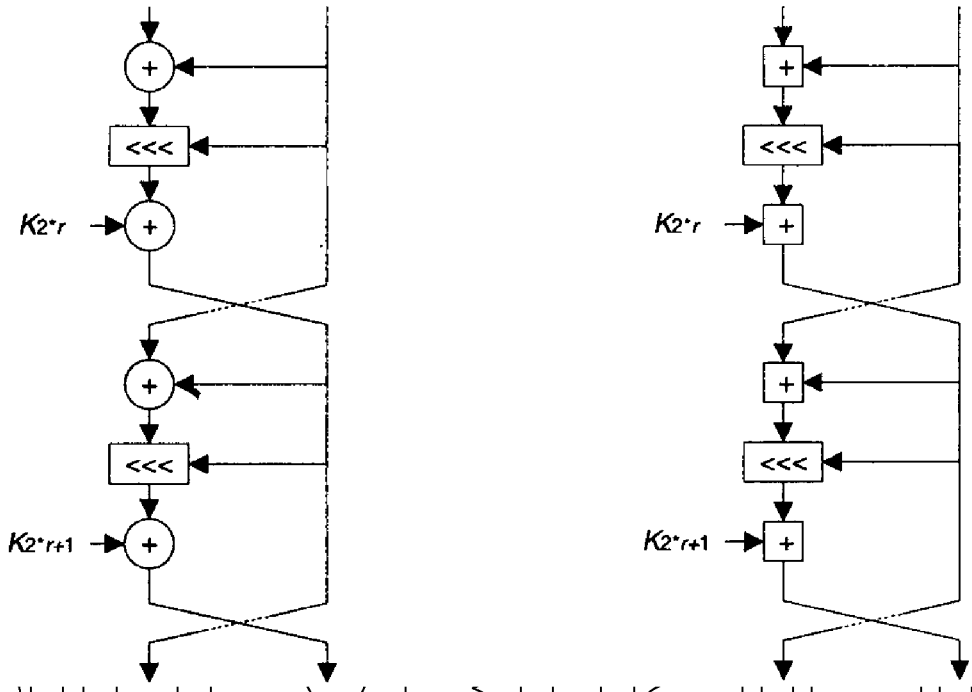

Рис.1.7. Раунди алгоритмів RC5XOR та RC5P

3. Алгоритм RC5PFR, який відрізняється від RC5 циклічним зрушенням на фіксоване, а не змінне число бітів (рис.1.8):

$$A = \left((A \oplus B) \lll s_r\right) + K_{2*r} mod 2^w, \quad (1.16)$$

де $s_r s_1 \dots s_R$ - число бітів циклічного зсуву, яке може бути різним у різних раундах алгоритму; у разі послідовність є додатковим параметром алгоритму.

Даний варіант алгоритму RC5 не є добре вивченим, проте експерти припускають, що алгоритм RC5PFR нестійкий проти диференціального криптоаналізу.

4. Алгоритм RC5KFR, у якому число бітів зсуву є функцією ключа шифрування KC, тобто, для кожного ключа шифрування число бітів зсуву є фіксованим (можливо різним для різних раундів алгоритму) (рис.1.8):

$$A = \left((A \oplus B) \lll s_r(KC)\right) + K_{2*r} mod 2^w. \quad (1.17)$$

Алгоритм RC5KFR також не є добре вивченим алгоритмом, проте вважається, що у багатьох випадках (особливо при недостатньо великій кількості раундів) криптоаналіз даного варіанту алгоритму RC5 зводиться до аналізу алгоритму RC5PFR, що не вселяє впевненості у його стійкості [7].

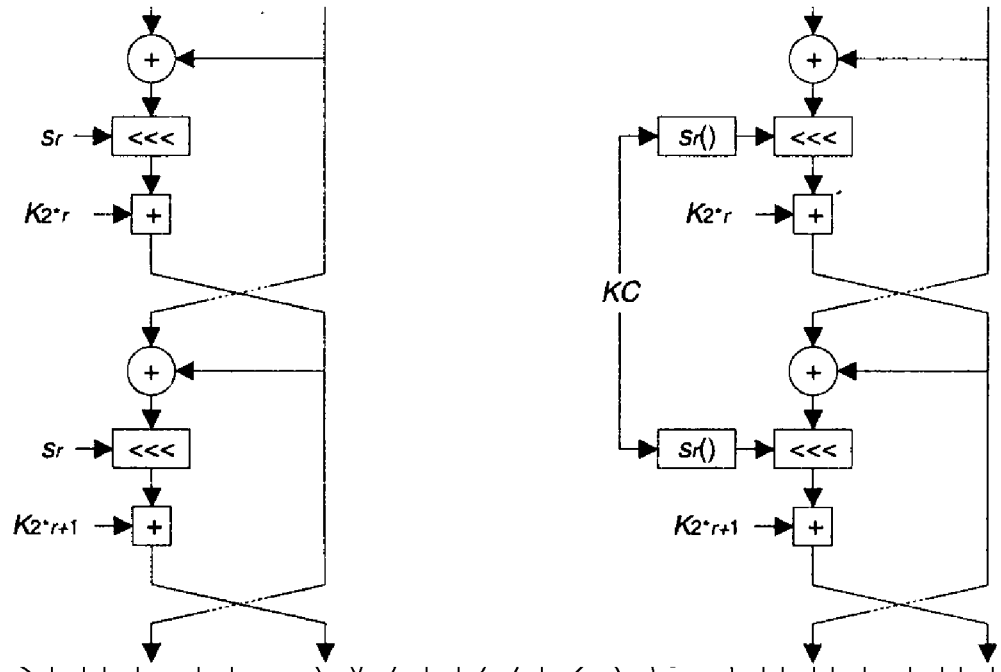

Рис.1.8. Раунди алгоритмів RC5PFR та RC5KFR

5. Алгоритм RC5RA, в якому виконується циклічний зсув на змінну кількість бітів, що визначається не значенням молодших бітів $log_2 w$ іншого субблоку, а якоюсь функцією f(), що обробляє як вхідне значення всі біти іншого субблоку:

$$A = \left((A \oplus B) \ll f(B)\right) + K_{2*r} mod 2^w. \qquad (1.18)$$

На основі алгоритму RC5 у 1998 р. було розроблено алгоритм RC6, який також заснований на циклічних зрушеннях на змінну кількість бітів. Переважна більшість криптоаналітичних досліджень алгоритму RC5 можуть бути різною мірою застосовані і до RC6.

Алгоритм RC6 був розроблений у 1998 р. рядом фахівців наукового підрозділу найвідомішої фірми RSA Data Security — RSA Laboratories: Рональдом Рівестом, Метом Робшоу, Реєм Сідні та Ікван Лайзою Ін спеціально для участі у конкурсі AES.

Алгоритм багато в чому успадкував риси попереднього алгоритму Рональда Рівеста - 64-бітного блокового шифру RC5. Фактично алгоритм зазнав двох принципових змін:

- на відміну від RC5, в алгоритмі використовується множення тільки за модулем 2^{32};

- для збереження 32-бітових обчислень замість розбиття шифрованого блоку даних (128 бітів згідно з принциповою вимогою конкурсу AES) на два 64-бітові субблоки виконується його розбиття на 4 32-бітових субблоки та їх обробка за дещо зміненою схемою.

Як і RC5, алгоритм RC6 має гнучку структуру: крім секретного ключа параметрами є:

- розмір w; алгоритм RC6 шифрує блоками по 4 слова;

- кількість раундів R;

- розмір секретного ключа у байтах b.

Аналогічно RC5 для уточнення параметрів алгоритму, що використовуються в його конкретній реалізації, застосовується позначення RC6-w/R/b.

Оскільки в конкурсі AES 128-бітний блок був обов'язковим, значення w фіксовано і 32. У специфікації алгоритму зафіксовано також кількість раундів: R=20.

Оскільки конкурс AES допускав використання ключів трьох фіксованих розмірів, розглянемо три наступні варіанти алгоритму: RC6-32/20/16, RC6-32/20/24 та RC6-32/30/32, сукупність яких ми й матимемо далі на увазі, говорячи про «RC6».

Структура алгоритму наведена на рис.1.9. В алгоритмі використовується 20 раундів перетворень, перед якими виконується часткове вхідне відбілювання:

$$B = B + K_0 \ mod \ 2^{32}, \quad (1.19)$$

$$D = D + K_1 \ mod \ 2^{32}, \quad (1.20)$$

де B і D – поточні значення двох із чотирьох (A, B, C, D) оброблюваних 32-бітових субблоків,

K_0 і K_1 - перші два з використовуваних в алгоритмі фрагментів розширеного ключа $K_0 \dots K_{43}$.

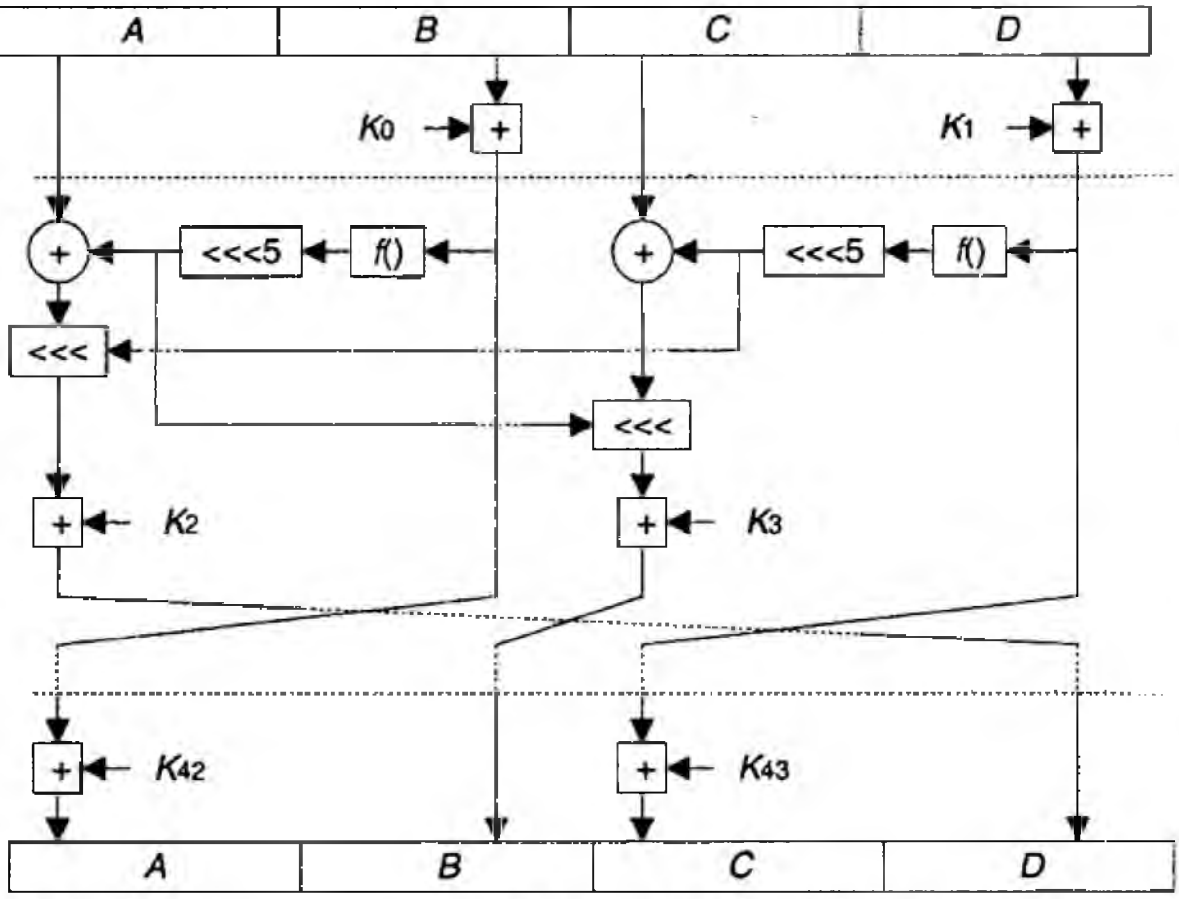

Рис.1.9. Структура алгоритму RC6

Аналогічним чином виконується часткове вихідне відбілювання.

$$A = A + K_{42}\ mod\ 2^{32}, \tag{1.21}$$

$$C = C + K_{43}\ mod\ 2^{32}. \tag{1.22}$$

У кожному i-му раунді алгоритму виконуються такі дії:

$$t_1 = f(B) \lll 5, \tag{1.23}$$

$$t_2 = f(D) \lll 5, \tag{1.24}$$

$$A = \left((A \oplus t_1) \lll t_2\right) + K_{2i}\ mod\ 2^{32}, \tag{1.25}$$

$$C = \left((C \oplus t_2) \lll t_1\right) + K_{2i+1}\ mod\ 2^{32}, \tag{1.26}$$

де t_1, t_2 і – тимчасові змінні.

Кількість бітів обертання на змінну кількість бітів визначається значенням 5 молодших бітів параметра (t_1 і t_2).

Функція f() виконує наступне квадратичне перетворення:

$$f(x) = x * (2x + 1) mod\ 2^{32} \tag{1.27}$$

Наприкінці кожного раунду виконується зсув субблоків.

При розшифровуванні підключі використовуються у зворотному порядку, накладання підключів замість додавання по модулю 2^{32} виконується відніманням, а також зсув субблоків виконується на початку раунду та у зворотний бік [8].

AES він же Rijndael - порівняно новий алгоритм шифрування, визнаний стандартом у США. Так само є блоковим шифром, а ось довжина ключів варіюється, може бути і 128 біт, і 196 біт, і навіть 256 біт, а блок вхідних даних має фіксовану довжину -128 біт.

На даний момент, це один із найпоширеніших алгоритмів шифрування. Даний алгоритм шифрування є одним з найнадійніших і використовується навіть для відомостей під грифом "Top Secret".

Шифрування за допомогою алгоритму AES здійснюється за допомогою кількох функцій:

- ExpandKey — функція для обчислення всіх раундових ключів;
- SubBytes — Функція для встановлення байтів, що використовує таблицю підстановок;
- ShiftRows - Функція, що забезпечує циклічний зсув в формі різні величини;
- MixColumns - Функція, яка змішує дані всередині кожного стовпця форми;
- AddRoundKey — Додавання ключа раунду з формою.

Процес шифрування зображений на рис.1.10.

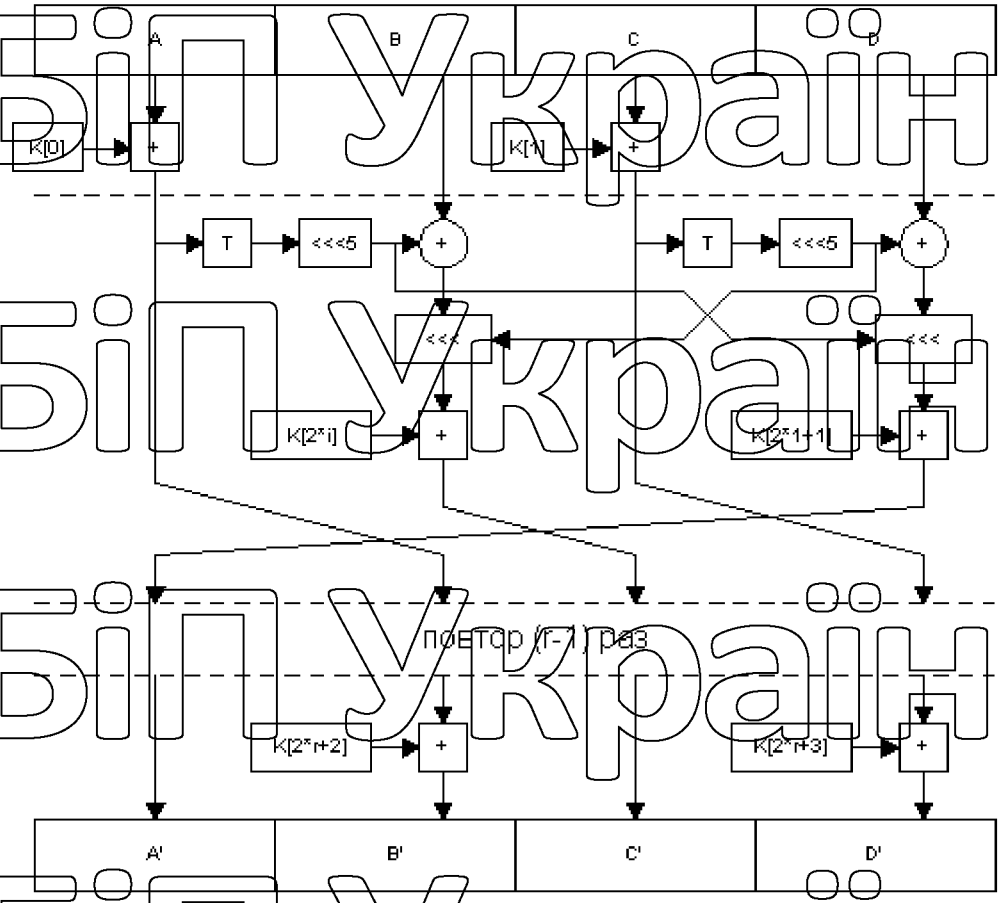

Рис.1.10. Процес шифрування за допомогою алгоритму AES

До AES широко використовувався DES, який також є офіційним стандартом із 1977 року. DES заснований на мережі Фейстеля з 16 циклами і ключем довжиною 56 біт, і розміром блоку 64 біта. Вхідними даними для блокового шифру служать:

- блок розміром n біт,
- ключ розміром k біт.

Принцип роботи:

1. Вихідні дані розбиваються на блоки фіксованої довжини (як правило кратиі ступеню двійки - 64 біт, 128 біт). У разі, якщо довжина блоку вихідних даних менша за довжину розрядності шифру, то блок доповнюється будь-яким заздалегідь відомим чином.

Блок ділиться на два рівні підблоки — «лівий» L_0 і «правий» R_0. У разі 64-бітної розрядності - на два блоки з довжиною 32 біти кожен.

3. «Лівий підблок» L_0 видозмінюється функцією ітерації $F(L_0, P_0)$ залежно від ключа P_0, після чого він складається за модулем 2 (XOR) з «правим підблоком» R_0.

4. Результат додавання присвоюється новому лівому підблоку L_1, який стає лівою половиною вхідних даних для наступного раунду, а «лівий підблок» L_0 надається без змін новому правому підблоку R_1, який стає правою половиною.

5. Ця операція повторюється n-1 раз, причому при переході від одного етапу до іншого змінюються раундові ключі (P_0, P_1, P_2 і т.д.), де n - кількість раундів для алгоритму.

Структурна схема алгоритму наведена на рис.1.11.

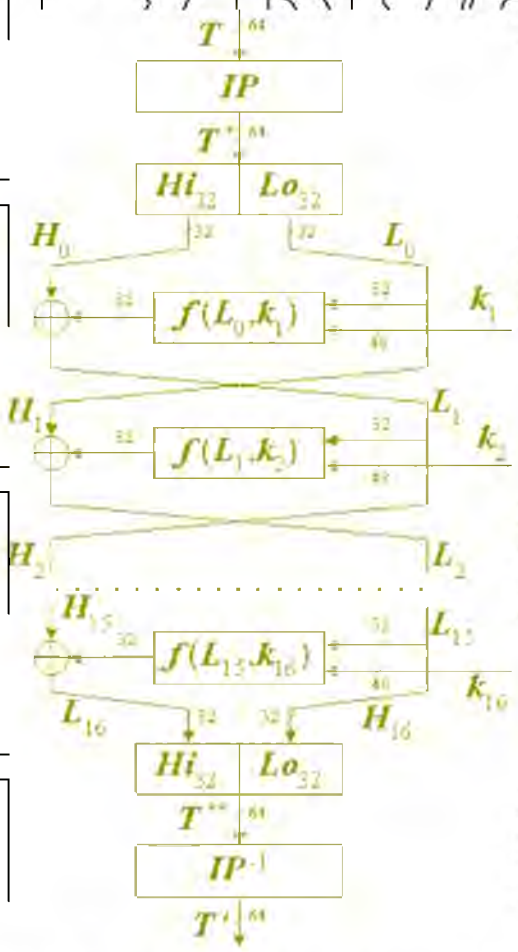

Рис.1.11. Структурна схема алгоритму DES

Процес розшифрування аналогічний до процесу шифрування за винятком того, що раундові ключі використовуються у зворотному порядку.

На виході (після застосування шифруючих перетворень) виходить зашифрований блок розміром n біт, причому незначні відмінності вхідних даних зазвичай призводять до суттєвої зміни результату.

Вся операція шифрування кожного блоку по 64 біта може бути розділена на 3 етапи [9]:

- початкова підготовка блоку даних;
- 16 раундів "основного циклу";
- кінцева обробка блоку даних.

Також відомий алгоритм 3DES. Блок-схема алгоритму наведена на рис.1.12

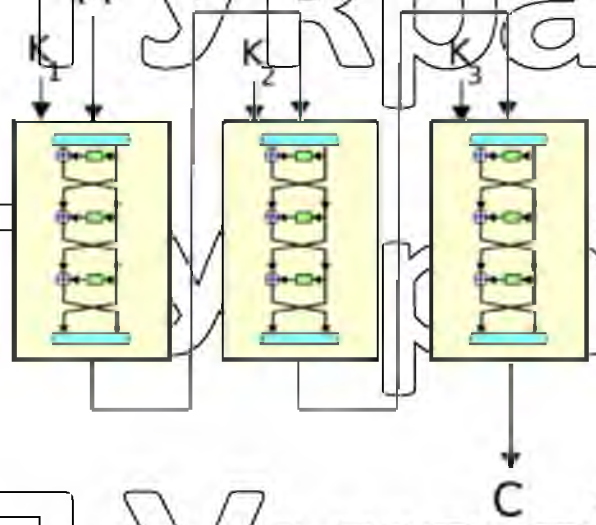

Рис.1.12. Спрощена схема алгоритму 3DES

Швидкість роботи 3DES в 3 рази нижче, ніж у DES, але криптостійкість набагато вище - час, необхідний для криптоаналізу 3DES, може бути в багато разів більше, ніж час, необхідний для розкриття DES. 3DES використовується частіше, ніж IDES, який легко ламається за допомогою сьогоднішніх технологій (наприклад, 1998 року організація Electronic Frontier Foundation, використовуючи спеціальний комп'ютер DES Cracker, розшифрувала DES за 3 дні). 3DES є простим способом усунення недоліків DES. Алгоритм 3DES побудований на основі DES, тому для його реалізації можна використовувати програми, створені для DES.

Далі розглянемо асиметричні алгоритми шифрування.

Алгоритм RSA. Названий на честь його розробників Рівеста(Ron Rivest), Шаміра (Adi Shamir) та Адлемана (Leonard Adleman), цей шифр досі є одним з найбільш широко використовуваних.

Шифр Шаміра повністю вирішує завдання обміну повідомленнями, закритими для прочитання, у разі, коли абоненти можуть скористатися лише відкритими лініями зв'язку. Однак, при цьому повідомлення пересилається три рази від одного абонента до іншого, що є недоліком. Шифр Ель-Гамаля дозволяє вирішити те саме завдання за одне пересилання даних, але обсяг шифротексту, що передається, в два рази перевищує обсяг повідомлення.

Система RSA позбавлена таких недоліків. Цікаво те, що вона базується на іншій односторонній функції, яка відрізняється від дискретного логарифму. Крім того, варто розглянути ще один винахід криптографії – одностороню функцію з лазівкою (trapdoor function).

Алгоритм роботи алгоритму RSA наведений на рисунку 1.13.

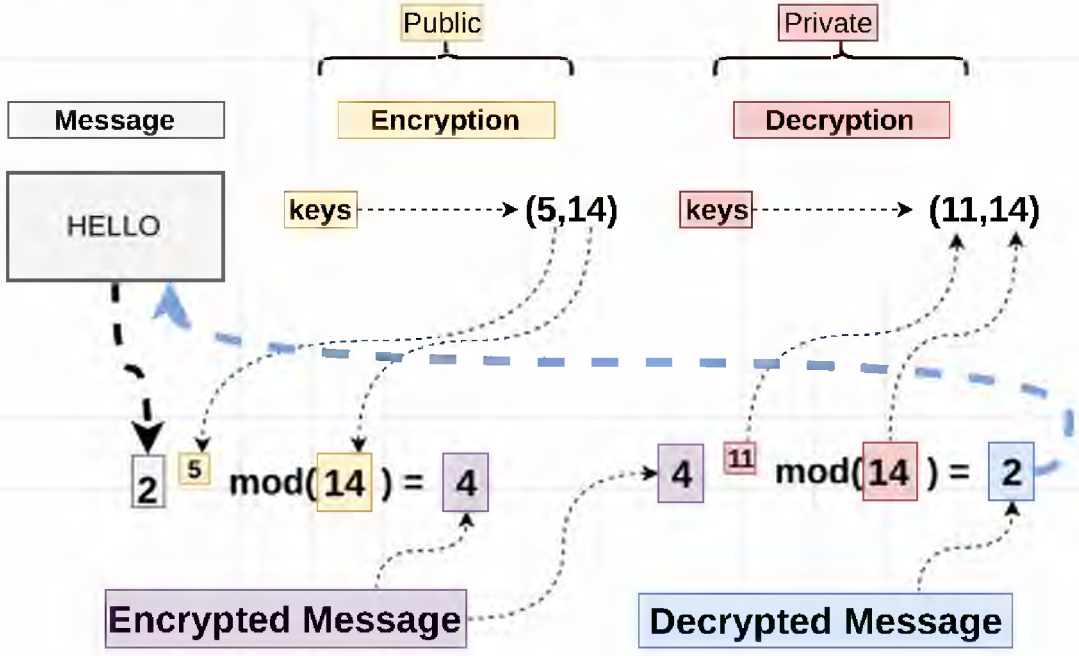

Рис. 1.13. Алгоритм роботи алгоритму RSA [10]

Ця система базується на наступних двох фактах з теорії чисел:

- Завдання перевірки числа на простоту є порівняно легким;

- завдання розкладання чисел виду n = pq (p і q - прості числа) на множники є дуже важким, якщо ми знаємо тільки n, а p і q - великі числа (це так зване завдання факторизації).

Нехай у системі є абоненти A, B, C, ... Кожен абонент вибирає випадково два великі прості числа P і Q. Потім він обчислює число

$$N = PQ. \quad (1.28)$$

(Число N є відкритою інформацією, доступною іншим абонентам.)

Після цього абонент обчислює число $\varphi = (P - 1)(Q - 1)$ і вибирає деяке число $d < \varphi$, взаємно просте з φ, і за узагальненим алгоритмом Евкліда знаходить число c, таке, що

$$cd \bmod \varphi = 1. \quad (1.29)$$

Ключі користувачів у системі RSA наведені у вигляді таблиці 1.3.

Таблиця 1.3 - Ключі користувачів у системі RSA

Абонент	Секретний ключ	Відкритий ключ
A	cA	d_A, N_A
B	cB	d_B, N_B
C	cC	d_C, N_C

Вся інформація, пов'язана з абонентами і є відкритими і секретними ключами.

Опишемо протокол RSA. Нехай Аліса хоче передати повідомлення m Бобу, причому повідомлення m розглядається як число, яке задовольняє нерівність $m < N_B$ (далі індекс B вказує на те, що відповідні параметри належать Бобу).

Крок 1. Аліса шифрує повідомлення за формулою:

$$e = m^{dB} \bmod NB, \quad (1.30)$$

використовуючи відкриті параметри Боба і пересилає e по відкритій лінії.

Крок 2. Боб, який отримав зашифроване повідомлення, обчислює

$$m^0 = e^{cB} \bmod N_B. \quad (1.31)$$

Для описаного протоколу $m^0 = m$, тобто абонент B одержує вихідне від A повідомлення.

При побудові протоколу $m_0 = e_{cB} \bmod N_B = m_{dBcB} \bmod N_B$.

Рівність (1.29) означає, що для деякого k:

$$c_B d_B = k\varphi_B + 1$$

$$\varphi_B = (P_B - 1)(Q_B - 1) = \phi(N_B), \qquad (1.32)$$

де $\phi(\cdot)$ – функція Ейлера.

Властивості протоколу RSA:

- Протокол шифрує та дешифрує інформацію коректно;

- Зловмисник, що перехоплює всі повідомлення і знає всю відкриту інформацію, не зможе знайти вихідне повідомлення за великих P і Q.

Для доказу другої властивості зауважимо, що зловмисник знає лише відкриті параметри N та d. Для того щоб знайти c, він повинен знати значення $\varphi = (P - 1)(Q - 1)$, а для цього, у свою чергу, йому потрібно знати P та Q.

Взагалі кажучи, він може знайти P та Q, розклавши N на множники, проте це складне завдання. Зазначимо, що вибір великих випадкових P і Q можливий за прийнятний час.

Одностороння функція $y = x^d \bmod N$, що застосовується в системі RSA, має так звану «лазівку», що дозволяє легко обчислити зворотну функцію $x = d^y \bmod N$ якщо відомо розкладання N на прості множники. (Дійсно, легко обчислити $\varphi = (P - 1)(Q - 1)$, а потім $c = d - 1 \bmod \varphi$.)

Якщо P та Q невідомі, то обчислення значення зворотної функції практично неможливо, а знайти P та Q по N дуже складно, тобто, знання P і Q - це «лазівка» або «потаємний хід»). Такі односторонні функції з лазівкою знаходять застосування та інших розділах криптографії.

Зазначимо, що з схеми RSA важливо, щоб кожен абонент вибирав власну пару простих чисел P і Q, тобто, всі модулі N_A, N_B, N_C, ... мають бути різні (інакше один абонент міг би читати зашифровані повідомлення, призначені іншого абонента).

Однак це не вимагається від другого відкритого параметра d. Параметр d може бути однаковим для всіх абонентів. Часто рекомендується вибирати d ∈ 3 (при відповідному виборі P і Q). Тоді шифрування виконується максимально швидко, лише за два множення.

Алгоритм Діффі-Хеллмана. Використовуючи функцію f(x) = x (mod p), вченими Діффі та Хеллманом була показана можливість побудови практично стійких секретних систем, які не вимагають передачі секретного ключа. Запропонована ними система отримала назву методу відкритого розповсюдження ключів.

У ній кожен абонент вибирає випадковим чином секретний ключ x і виробляє відкритий ключ y відповідний вибраному секретному ключу, відповідно до формули

$$y = \alpha^x \pmod p. \qquad (1.33)$$

Алгоритм наведений на рисунку 1.14 [11].

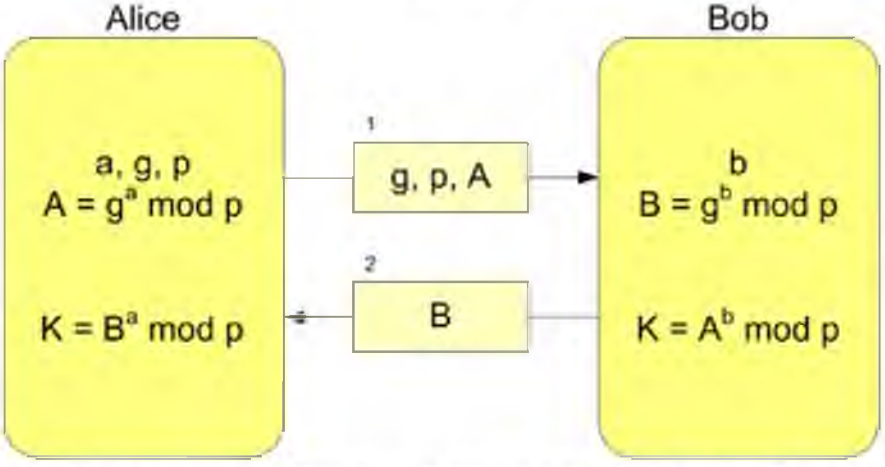

Рис. 1.14. Алгоритм Діффі-Хеллмана

Системою Діффі-Хеллмана називається наступний спосіб використання дискретного зведення у ступінь для обміну секретними ключами між користувачами мережі із застосуванням лише відкритих повідомлень.

Вибирається велике просте число p та відповідний йому корінь a <p. Для забезпечення стійкості розглянутої системи відкритого шифрування на число p накладається така умова: розкладання числа p-1 на множники повинно

містити принаймні один великий простий множник; розмір числа p має бути не менше 512 біт.

Механізм розподілу секретних ключів по відкритому каналу полягає у наступному. Кожен абонент вибирає випадковий секретний ключ x і виробляє відкритий ключ y, що відповідає обраному секретному ключу, відповідно до формули

$$y = x \pmod p. \qquad (1.34)$$

Два абоненти можуть встановити секретний зв'язок без передачі секретного ключа наступним чином. Абонент А бере відкритий ключ у абонента В і, використовуючи свій секретний ключ xA, обчислює загальний секретний ключ:

$$Z_{AB} = (y_B)^{X_A} = (\alpha^{X_B})^{X_A} = \alpha^{X_B X_A} \pmod p. \qquad (1.35)$$

Аналогічно для другого абонента:

$$Z_{BA} = (y_A)^{X_B} = (\alpha^{X_A})^{X_B} = \alpha^{X_B X_A} \pmod p. \qquad (1.36)$$

Таким чином, обидва абоненти сформували однаковий секретний ключ ZAB без використання будь-якого заздалегідь обумовленого загального секрету.

Володіючи тільки їм відомим секретом і використовуючи його як майстер-ключ, дана пара абонентів може зашифровувати повідомлення, що направляються один одному.

Зазначені вище обчислення легко здійсненні для досить великих значень p, a, у них (наприклад, що мають у двійковому поданні довжину 4096 біт і більше).

Атакуючому відомі значення $y_B = \alpha^{X_B} \pmod p$ і $y_A = \alpha^{X_A} \pmod p$, але для того, щоб обчислити ZAB, він повинен вирішити задачу дискретного логарифмування і визначити або xA, або xB.

Якщо буде знайдено обчислювально ефективні методи розв'язання задачі дискретного логарифмування, то метод Діффі-Хеллмана виявиться неспроможним — у зв'язку з цим говорять, що даний метод відкритого

розподілу ключів заснований на складності дискретного логарифмування. В даний час у загальному випадку завдання дискретного логарифмування практично нерозв'язне, що дає можливість широкого практичного застосування методу Діффі-Хеллмана та численних систем ЕЦП, заснованих на складності обчислення дискретних логарифмів.

Проте варто відмітити, що протокол обміну ключами Діффі-Хеллмана не забезпечує автентичності узгодженого ключа. Активний противник, впроваджений в канал зв'язку між двома користувачами А і Б, може маніпулювати повідомленнями протоколу і розпочати атаку “людина посередині” (man-in-the-middle attack).

У ході цієї атаки Зловмисник перехоплює і блокує перше повідомлення від А до Б, тобто число ga маскується під А і посилає Б наступне повідомлення.

Зловмисник (під ім'ям А) – Б: gm = gm (mod p).

Б, дотримуючись інструкцій протоколу, повертає Зловмиснику (під ім'ям А) число gb. Це число знову перехоплюється та блокується Зловмисником. Тепер Зловмисник і Б узгодили між собою ключ gb m =(mod p), хоча Б вважає, що він поділив цей ключ з А.

Аналогічно Зловмисник, імітуючи Б, може узгодити інший ключ з А. Згодом зловмисник може використовувати обидва ключі для читання та заміни “секретних” повідомлень, якими обмінюються А та Б, або по черзі імітувати цих користувачів.

Атака на протокол обміну ключами Діффі-Хеллмана є цілком реальною, оскільки цей протокол не передбачає перевірки автентичності джерела повідомлень. Для того щоб ключі були узгоджені лише між А та Б, обидві сторони мають бути впевнені одна в одній.

Мінуси програмної реалізації [11], що знижують стійкість:

- при обчисленні функції step(), з'являється пряма залежність часу обчислення кількості одиниць (при бінарному поданні) в степені. Це може дати уявлення про кількість одиниць у секретному ключі.

- у програмному продукті не реалізовано повне очищення оперативної пам'яті.

- секретні ключі зберігаються у відкритому вигляді на диску.

- не реалізована можливість обчислення хеш, внаслідок чого можлива модифікація ключів.

ECC. Це алгоритм асиметричного шифрування нового покоління. Він також використовується для створення ключів шифрування та створення безпечних з'єднань для безпечної передачі даних. ECC набагато швидше та безпечніше, ніж RSA або DSA. Він використовує коротші ключі, ніж RSA, які так само складно зламати. Наприклад, 512-бітовий ECC ключ так само безпечний, як і 15360-бітний ключ RSA, але оскільки він набагато коротший, то споживає значно менше обчислювальної потужності для його генерації. ECC використовується не так часто, як RSA, оскільки він відносно новий, до того ж, RSA легше реалізувати [12].

Еліптична крива криптографії (ECC) існує з середини 1980-х років, але вона, як і раніше, розглядається як новачок у світі SSL і тільки почала отримувати визнання за останні кілька років. ECC - принципово інший математичний підхід до шифрування, ніж алгоритм RSA. Еліптична крива є функцією алгебри ($y^2 = x^3 + ax + b$), яка виглядає як симетрична крива, паралельна осі x при побудові графіка.

Алгоритм шифрування наведений на рисунку 1.15.

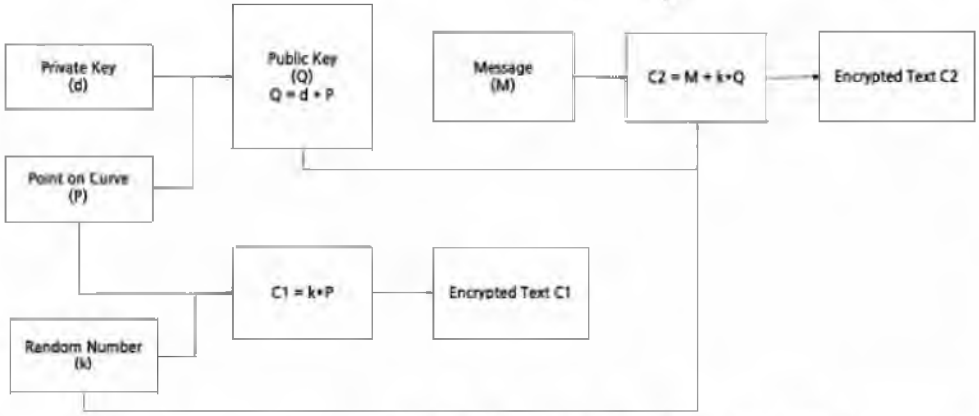

Рис. 1.15. Алгоритм шифрування ECC

Як і в інших формах криптографії з відкритим ключем, ECC заснований на односторонній властивості, в якій легко виконати обчислення, але система не піддається зворотному результату розрахунку, щоб знайти вихідні числа. Для досягнення цієї властивості ECC використовує інші математичні операції, ніж RSA. Найпростіший спосіб пояснити цю математику - для еліптичної кривої лінія проходитиме тільки через три точки вздовж кривої (P, Q і R) і що, знаючи дві з точок (P і Q), інша (R) можна легко обчислити, але тільки з R, дві інші, P і Q, не можуть бути отримані.

ECC використовується як у цифрових підписах за допомогою еліптичної кривої DSA (ECDSA), так і при обміні ключами через Elliptic Curve Diffie-Hellman (ECDH).

Ці алгоритми застосовуються у різних частинах стандарту SSL. По-перше, сертифікати SSL можуть бути підписані з ECDSA замість RSA. Друге використання для ECC, коли веб-сервер та клієнт узгоджують ключі сеансу, які використовуються для шифрування всіх даних, що надсилаються між сервером та браузером. У цьому останньому випадку сервер та браузер мають бути налаштовані для підтримки наборів шифрування ECDH.

Головною перевагою ECC є те, що він сильніший за RSA для ключових розмірів, що використовуються сьогодні.

Щоб випередити обчислювальну потужність зловмисника, ключі RSA повинні бути довшими.

Іншою перевагою ECC у плані безпеки є просте надання альтернативи RSA та DSA. Якщо виявлено серйозну слабкість RSA, ECC, ймовірно, стане кращою альтернативою, особливо якщо раптова слабкість RSA вимагає різкого збільшення розміру ключа для компенсації [12].

ECC також швидше з низки причин. По-перше, менші ключі означають менше даних, які мають бути передані із сервера клієнту під час встановлення зв'язку SSL. Крім того, ECC вимагає менше обчислювальної потужності (ЦП) та пам'яті, що призводить до значного збільшення часу відгуку та пропускної спроможності на веб-серверах, коли вони використовуються.

Порівняння алгоритмів шифрування наведене у вигляді таблиці 1.4 [13].

Таблиця 1.4 - Порівняння алгоритмів шифрування

Назва алгоритму	Структура	Гнучкість та можливість розширення	Атаки, до яких алгоритм вразливий
DES	Мережа Файстеля	Ні	Атака «грубою силою»
3DES	Мережа Файстеля	Так. Розширена від 56 до 168 біт	Атака «грубою силою», атака з відомим відкритим текстом, атака на основі підібраного відкритого тексту
CAST-128	Мережа Файстеля	Так, 128 та 256 біт	Атака на основі підібраного відкритого тексту
BLOWFISH	Мережа Файстеля	Так. Довжина ключа: 64-448 біт (кратні 32)	Перебір за словником
IDEA	SP-мережа	Ні	Атака по часу, атака розкладу ключів(key schedule attack)
AES	SP-мережа	Так. Довжина ключа: 256 біт(кратні 64),	Атака сторонніми каналами
RS6	Мережа Файстеля	Так. Довжина ключа: 128 – 2048 біт(кратні 32)	Атака «грубою силою», аналітична атака
RSA	Факторизація	Так. Multi-prime RSA, multi-power RSA	Факторинг публічного ключа

Варто відзначити, що безпека криптографічного алгоритму визначається тим, наскільки алгоритм стійкий до різних атак. Ефективність криптографічних алгоритмів базується на структурі, ключі довжині, розмірі

блоку, кількості виконаних ітерацій та часі виконання. Зрештою, все це є факторами, що впливають на безпеку певного алгоритму [13].

1.3 Постановка завдання щодо проведення магістерського дослідження

Згідно з інформацією, наведеною в [14], існує дві базові вимоги, які мають бути враховані при виборі системи шифрування, незалежно від стратегії безпеки, що використовується в організації, яка буде цю систему застосовувати.

1. Надійність системи Основне питання, що хвилює будь-якого користувача – наскільки надійна обрана система в принципі і наскільки її можливості відповідають вимогам безпеки даних, що висуваються інформаційною системою, яку він застосовує.

Надійна система має використовувати ефективний та швидкодіючий алгоритм шифрування.

2. Зручність системи. Друге по важливості питання, яке завжди хвилює користувача - наскільки використання обраної системи шифрування ускладнить виконання основної роботи в інформаційній системі і як її застосування позначиться на загальній продуктивності праці.

Зручна система ніяк не впливає на ефективність роботи, оскільки використовує метод так званого прозорого шифрування, коли вся інформація, з якою користувач працює, шифрується при записі, а при читанні - розшифровується.

Цей процес не вимагає участі користувача та не розпізнається стандартними програмами, які використовуються для роботи з цією інформацією. Отже, сам факт роботи із секретними даними може бути невідомий, що передбачає високий рівень надійності такого захисту.

Слід враховувати, що забезпечення безпеки та цілісності даних на одному-двох комп'ютерах та забезпечення надійного захисту кількох десятків

робочих станцій потребує принципово різного підходу. Якщо в першому випадку досить зашифрувати лише деякі файли, у другому великий обсяг використовуваної інформації вимагає часом шифрування цілих дискових масивів. Тому при виборі системи шифрування важливо враховувати наявність додаткових можливостей, що забезпечують гнучкість налаштувань з урахуванням індивідуальних завдань клієнта.

В межах виконання роботи буде проведена розробка та тестування системи швидкого шифрування дискретних даних.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ШИФРУВАННЯ ДИСКРЕТНИХ ДАНИХ

В межах даного розділу розглянемо алгоритм розробленої системи шифрування дискретних даних.

Вважається, що файл має структуру:

$$A_0, \dots A_n, \quad (2.1)$$

де A_0,..., A_n - байти (n>0).

При цьому виконане припущення:

1. Якщо B - байт, то N(B) - вміст байта у двійковій системі числення, що інтерпретується як ціле число (без знака).

2. rest(i,j) - залишок від розподілу цілого числа (без знака) i на натуральне j.

По визначенню:

$$rest(m,0) = m, rest(m,1) = 0, \quad (2.2)$$

для будь-якого значення m.

3. S_0, S_q – ключове слово, що складається з q+1 байтів S_0, ..., S_q (q>0),

4. Знак «*» – побітове додавання за модулем 2.

5. Послідовність C_0 … C_q C_{q+1} … C_{q+n+1} – це S_0…S_q A_0…A_n.

Розглянемо безпосередньо шифрування дискретних даних.

А) Пряма формула кодування.

(1) Початковий крок:

for i=0 to q step 1 do (C_i:=S_i);

for i=0 до n Step 1 do (C_{i+q+1}:=A_i);

r:=q+1; m=rest(N(C_{r-1}),r); C_r: = C_r * C_m;

(2) r:=r+1; if r >q+n+1 then STOP;

m:=rest(N(C_{r-1}),r);

$C_r = C_r * C_{r-1} * C_m$, goto (2);

Останнє можна переписати у вигляді:

$C_r := A_{r-q-1} * C_{r-1} * C_m$; goto (2);)

Послідовність, $C_{q+1} \dots C_{q+n+1}$ яка отримана після STOP – результат кодування.

Б) Зворотна формула кодування.

Для зворотної формули кодування припустимо, що $C_0 \dots C_{q+n+1}$ - послідовність байтів, що надходить. При цьому вважається що послідовність $C_0 \dots C_q$ відома і відомий ключ $S_0 \dots S_q$). При цьому $A_0 \dots A_n$ - вихідна послідовність байтів.

Нижче наведений алгоритм декодування.

(1) Початковий крок:

r:=q+1; m=rest(N(C_{r-1}),r); $A_{r-q-1} = C_r * C_m$;

(2) r:=r+1; if r >q+n+1 then STOP;

m:=rest(N(C_{r-1}),r);

$C_r := C_r * C_{r-1} * C_m$; goto (2);

Останнє можна переписати у вигляді:

$A_{r-q-1} := C_r * C_{r-1} * C_m$; goto (2);)

Послідовність, $A_0 \dots A_n$, яка отримана після STOP – результат декодування.

Проектування системи виконувалося завдяки середовищу Visual Studio.

Переваги обраного середовища:

1. Точне кодування У Visual Studio користувачам надається допомога у написанні коду в реальному часі, незалежно від того, яку мову програмування вони використовують. Вбудований у платформу IntelliSense пропонує підказки та описи API та автозаповнення рядків для підвищення швидкості.

Крім того, інтегроване середовище розробки Visual Studio гарантує, що розробники не втратять місце останньої модифікації щодо іншої частини коду.

2. Швидке налагодження. Пошук та діагностика помилок можуть бути складним завданням, але в Visual Studio є безліч інструментів, які спрощують це завдання. Платформа підтримує налагодження для всіх увімкнених мов. Процес також може виконуватися локально, віддалено або у процесі виробництва. Це дозволяє розгортати програми на робочому столі або емулятори на мобільних пристроях та використовувати інші методи налагодження.

3. В архітектурному плані Visual Studio Code поєднує в собі найкращу веб-технологію, нативні та мовні технології. Visual Studio Code поєднує в собі веб-технології, такі як JavaScript та Node.js, зі швидкістю та гнучкістю власних програм.

Крім того, Visual Studio Code використовує архітектуру сервісів інструментів, яка дозволяє йому інтегруватися з багатьма з тих же технологій, які використовуються у Visual Studio, включаючи Roslyn для .NET, TypeScript, механізм налагодження Visual Studio та інші [15].

РОЗДІЛ 3. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ШИФРУВАННЯ ДИСКРЕТНИХ ДАНИХ

3.1 Інтерфейс програми та інструкція користувача

В межах даного розділу роботи розглянемо результати дослідження інтелектуальної системи шифрування дискретних даних.

Виконаємо опис розробки. При запуску додатка перед користувачем відкривається головна сторінка з описом програми та невеликою інструкцією користувача (рисунок 3.1).

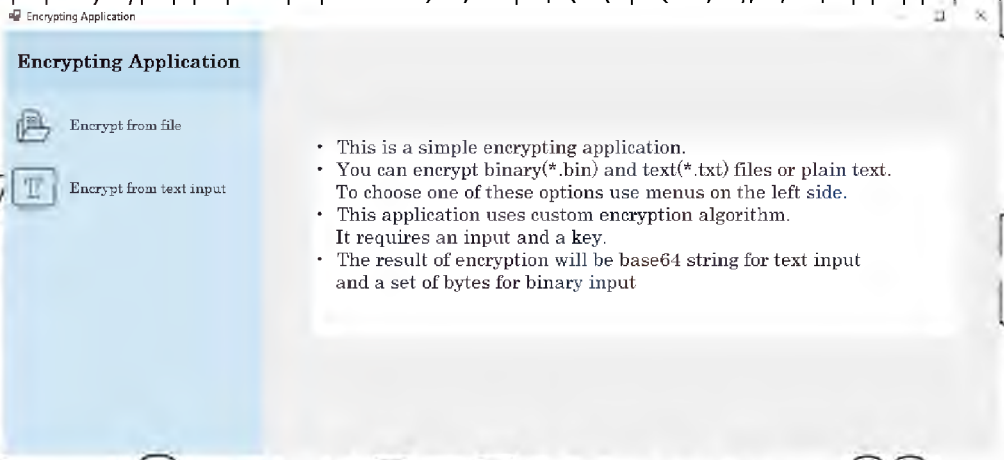

Рис.3.1. Інструкція користувача

Зліва від головного екрану знаходиться меню для вибору формату, у якому будуть подаватися вхідні дані – файлами або звичайним текстом (рисунок 3.2).

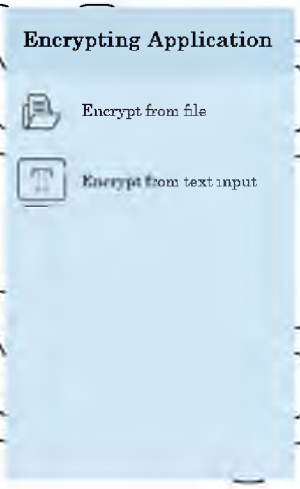

Рис.3.2. Меню для вибору формату

При виборі файлового формату вхідних даних перед користувачем відкривається екран вигляду, який представлений на рисунку 3.3.

Рис.3.3. Результат вибору файлового формату вхідних даних

Для початку шифрування користувач повинен обрати файл, який містить дані, які будуть шифруватися, файл з ключем шифрування, а також директорію, в яку буде збережено файл із зашифрованою інформацією.

Якщо вихідна директорія не вказана – за замовчуванням файл буде збережено по шляху D:\Output.txt(або D:\Output.bin). Попередньо потрібно обрати розширення файлів, з якими користувач буде працювати. Підтримувані розширення - *bin (бінарні файли) та *.txt(текстові файли) (рисунок 3.4).

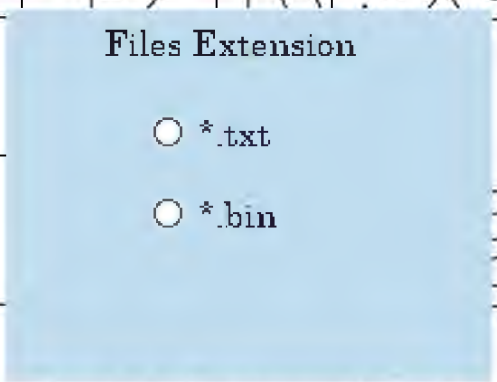

Рис.3.4. Розирення файлів

Після виконнаня всіх вищезазначених пунктів користувач має можливість розпочати процес шифрування вхідних даних, натиснувши кнопку «Encrypt» (рисунок 3.5).

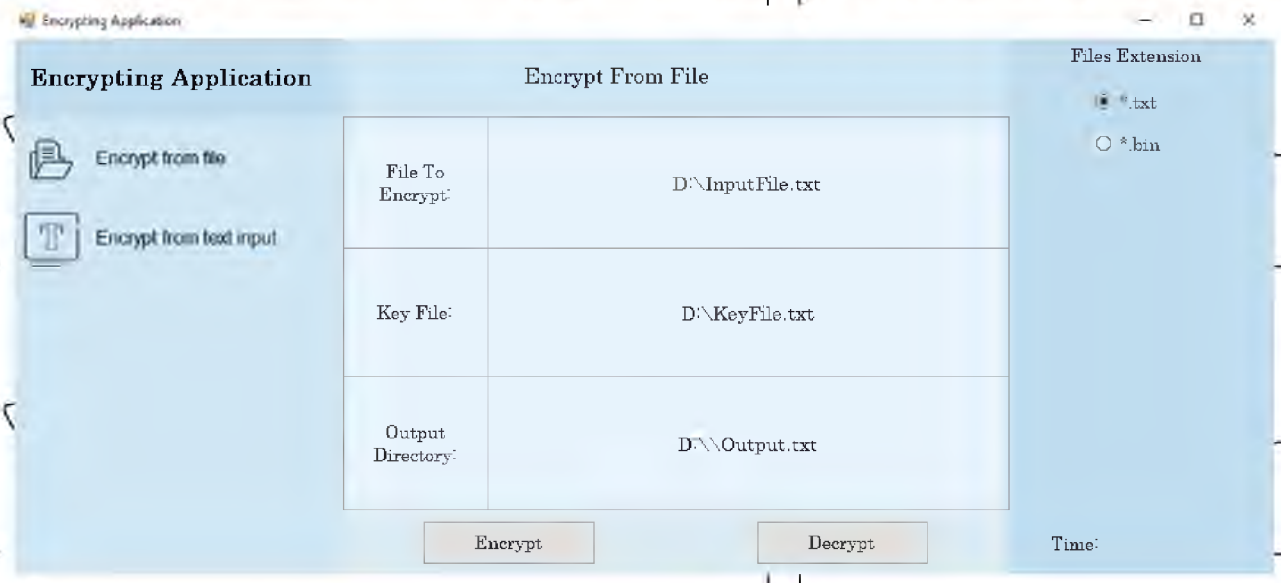

Рис.3.5. Можливість розпочати процес шифрування вхідних дани

Після закінчення шифрування на екран виводиться відповідне повідомлення та фіксується час роботи алгоритму в мілісекундах: (рисунок 3.6).

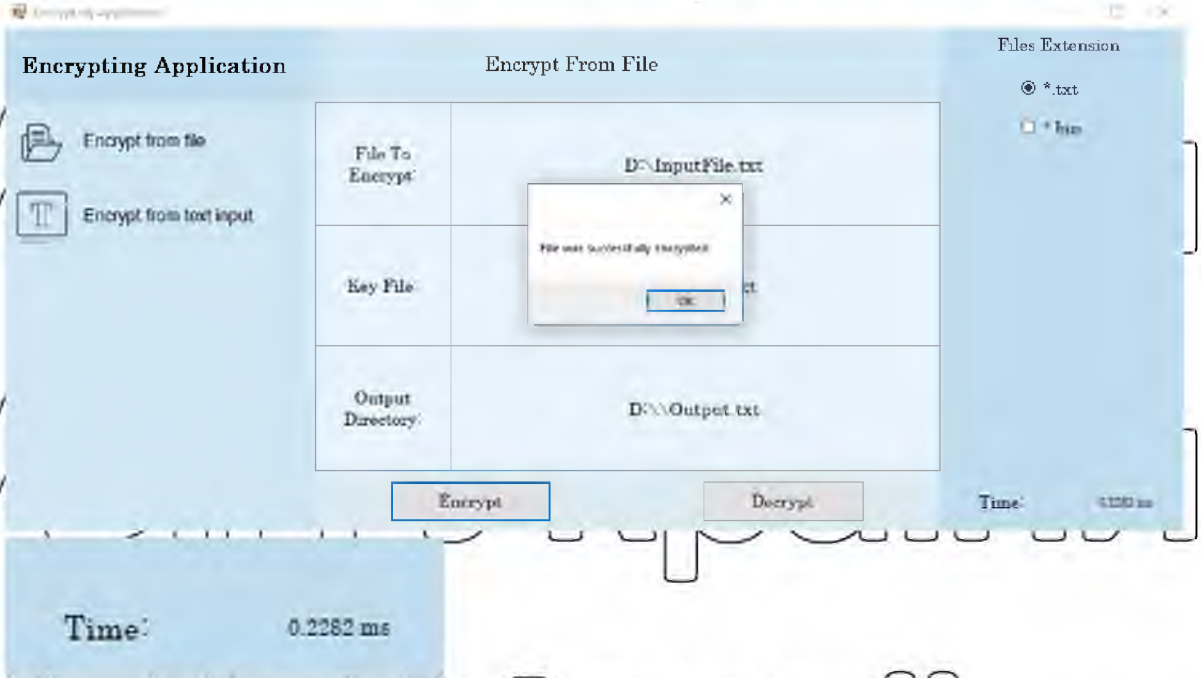

Рис.3.6. Вивід на екран часу шифрування

Процес дешифрування представляє собою зворотній процес, де на вхід програмі подається файл з зашифрованими даними, який за допомогою того ж ключа, що використовувався для шифрації, дешифрує інформацію назад до початкового вигляду (рисунок 3.7).

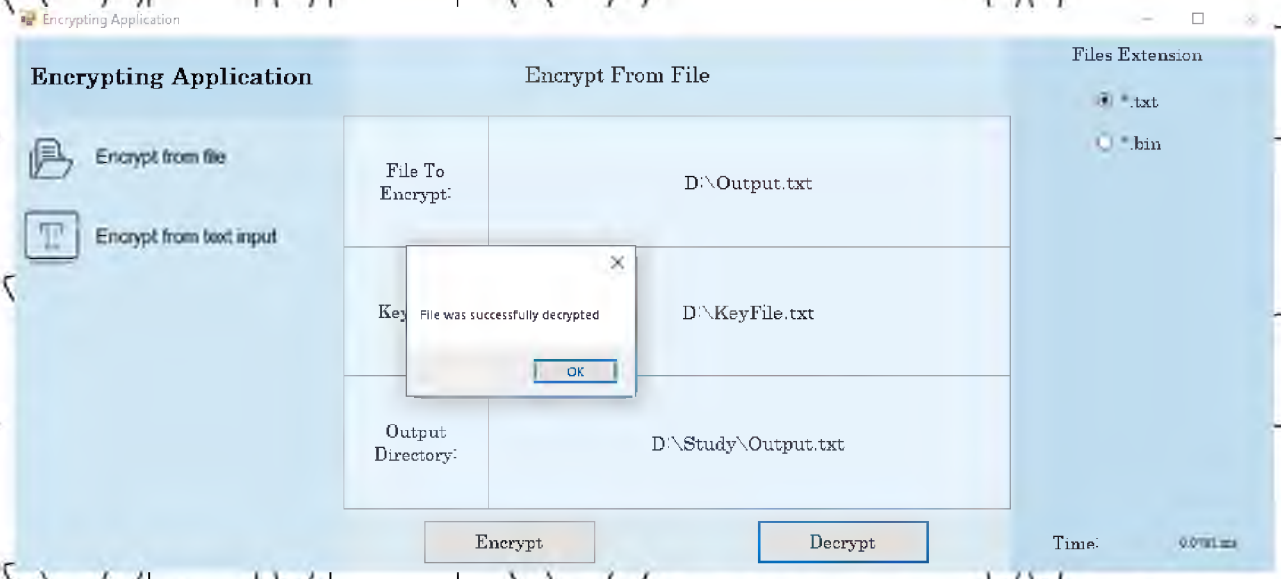

Рис.3.7. Інтерфейс вікна дешифрування дискретних даних

Варто відмітити, що якщо шифрування відбувається над текстовими даними, то вихідний файл задля уникнення проблем із кодуванням символів в різних системах(Unicode, UTF8 тощо) буде містити зашифрований текст у форматі base64.

При спробі натиснути кнопки «Encrypt» або «Decrypt», попередньо не вибравши файли, або при спробі вибрати файли, попередньо не вибравши їхнє розширення, спрацьовує валідація (рисунок 3.8).

Рис.3.8. Валідація

Панель, де вхідні дані записуються у текст бокси наведена на рисунку 3.9.

Рис.3.9. Панель, Панель, де вхідні дані записуються у текст бокси

Приклад виконання шифрування представлений на рисунку 3.10.

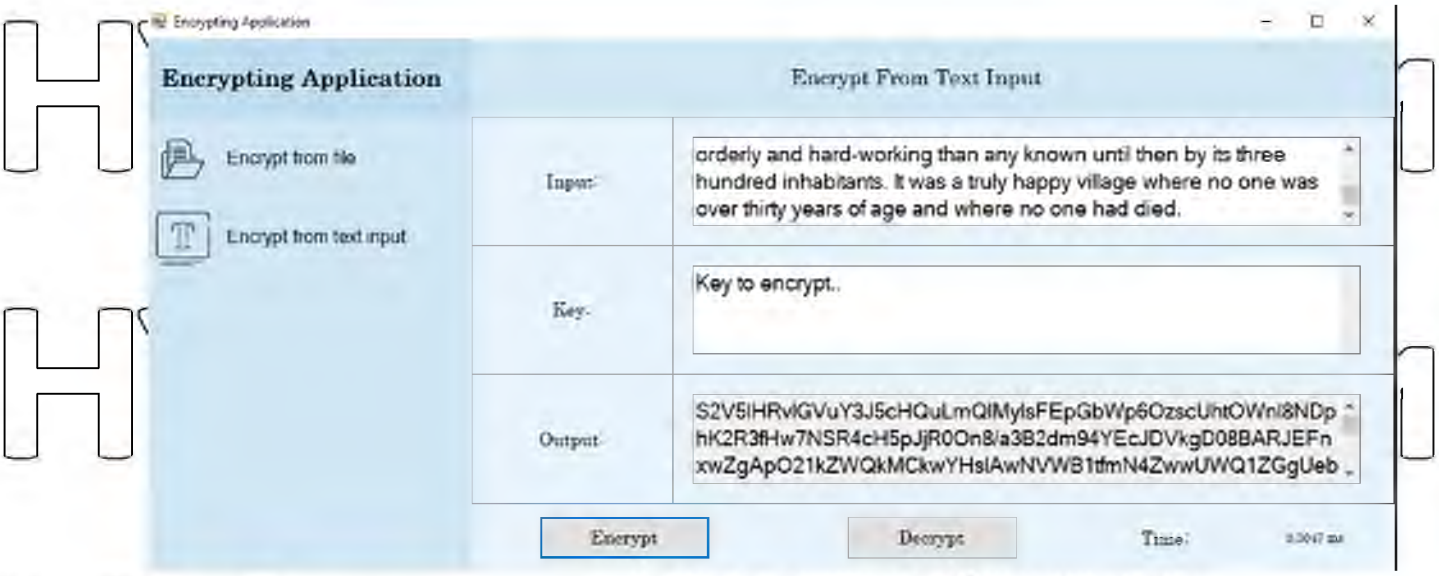

Рис.3.10. Приклад виконання шифрування

Приклад виконання дешифрування представлений на рисунку 3.11.

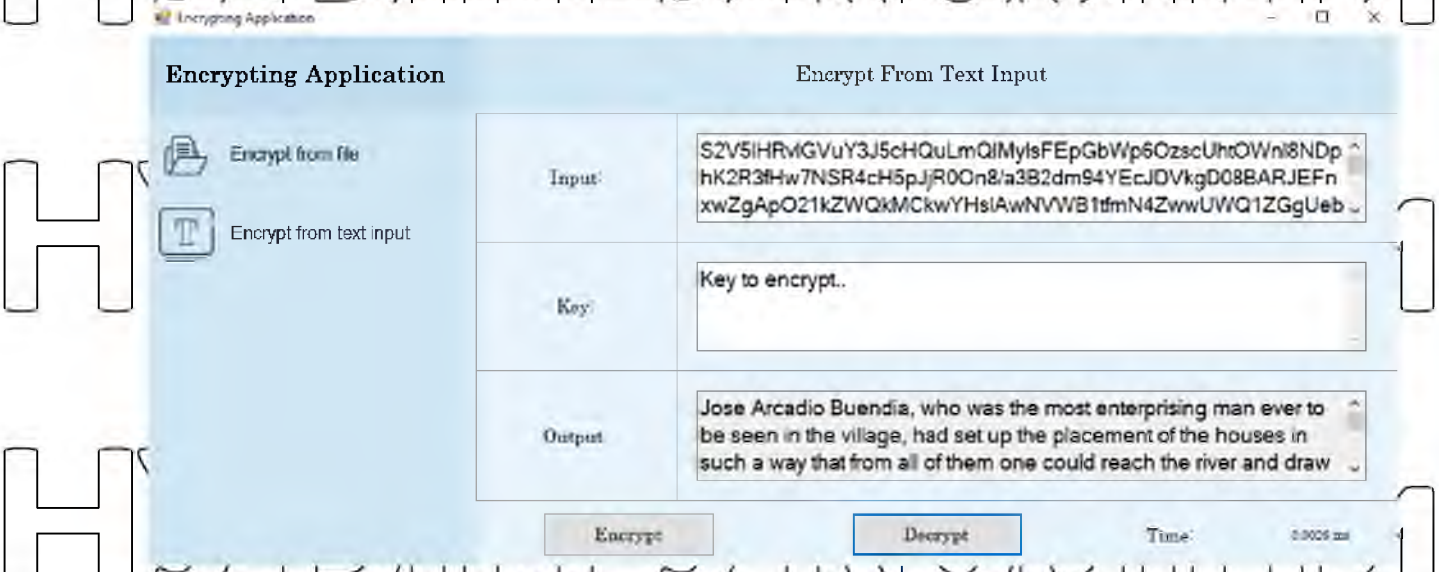

Рис.3.11. Приклад виконання дешифрування

При спробі виконати шифрацію або дешифрацію, не ввівши необхідні дані – спрацьовує валідація (рисунок 3.12).

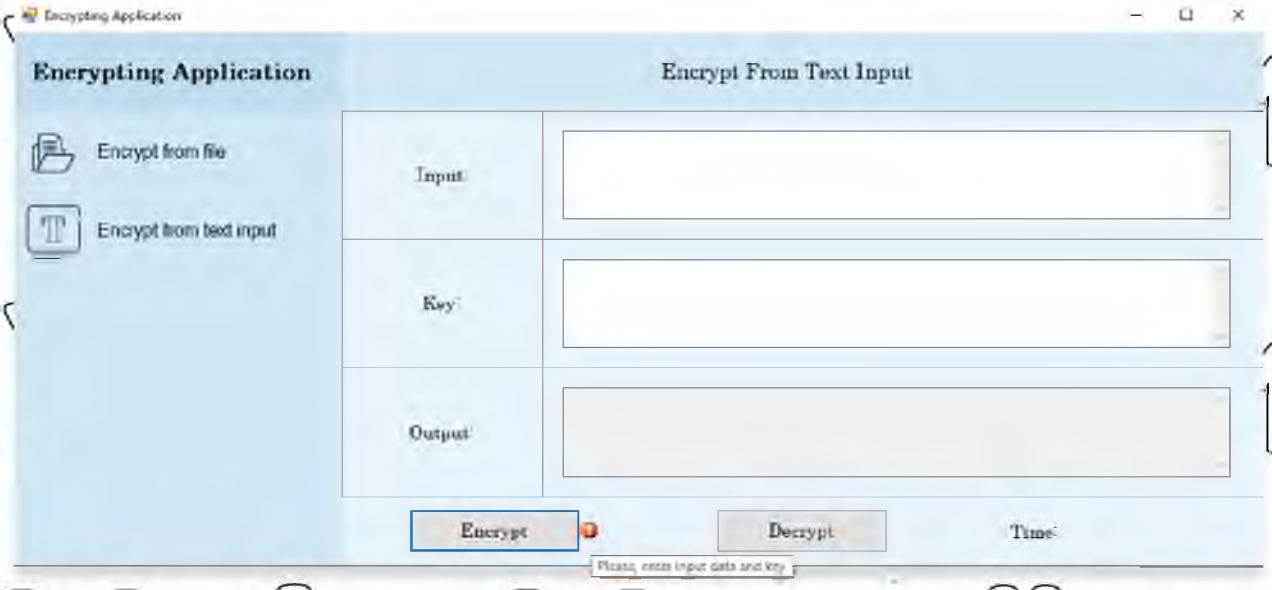

Рис.3.12. Валідація

Приклади реалізації будуть наведені далі.

3.2 Приклади реалізації

Розглянемо приклади шифрування та дешифрування дискретних даних для виявлення працездатності роботи інтелектуальної системи шифрування дискретних даних.

Приклад 1. Вхідний файл наведений на рис.3.13.

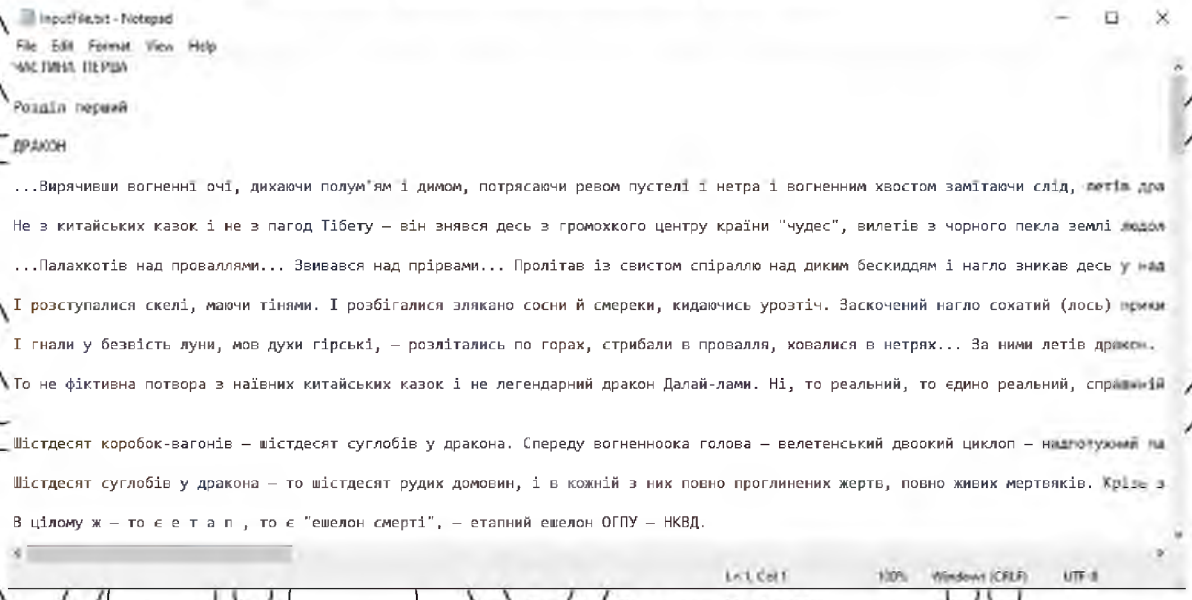

Рис.3.13. Вхідний файл

Ключ (128 біт) наведений на рисунку 3.14.

Рис.3.14 Ключ

Вихідний(зашифрований) файл наведений на рисунку 3.15.

Рис.3.15. Результат шифрування дискретних даних

Вихідний файл після дешифрування представлений на рис.3.16.

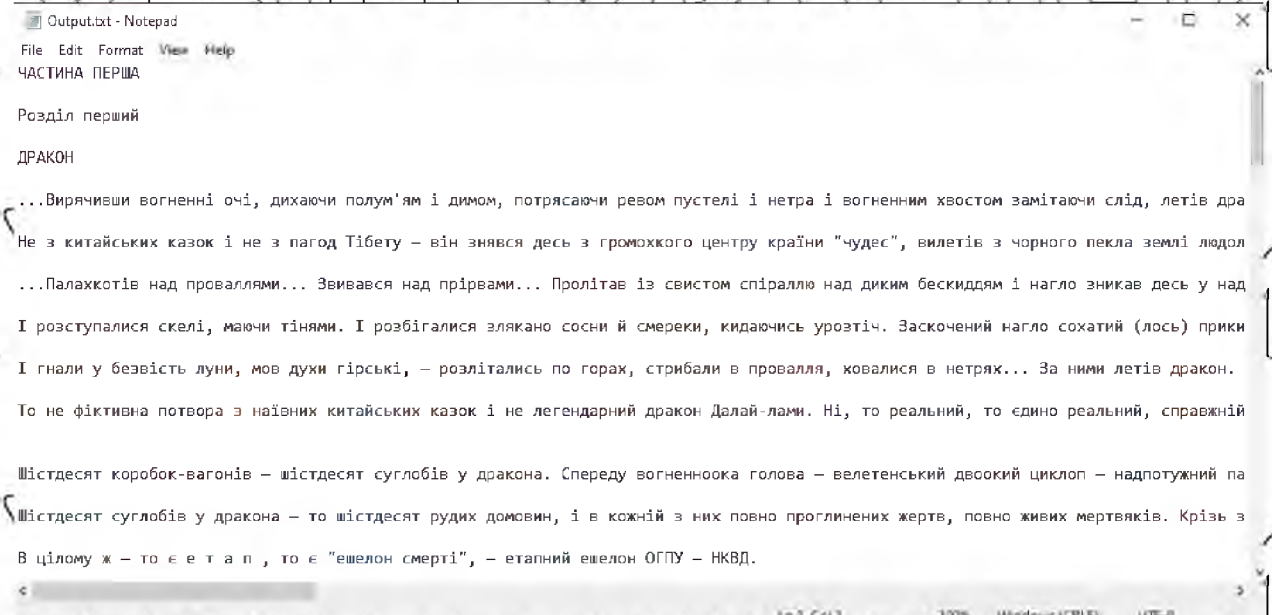

Рис.3.16. Вихідний файл після дешифрування

Приклад 2. Вхідний файл наведений на рис.3.17.

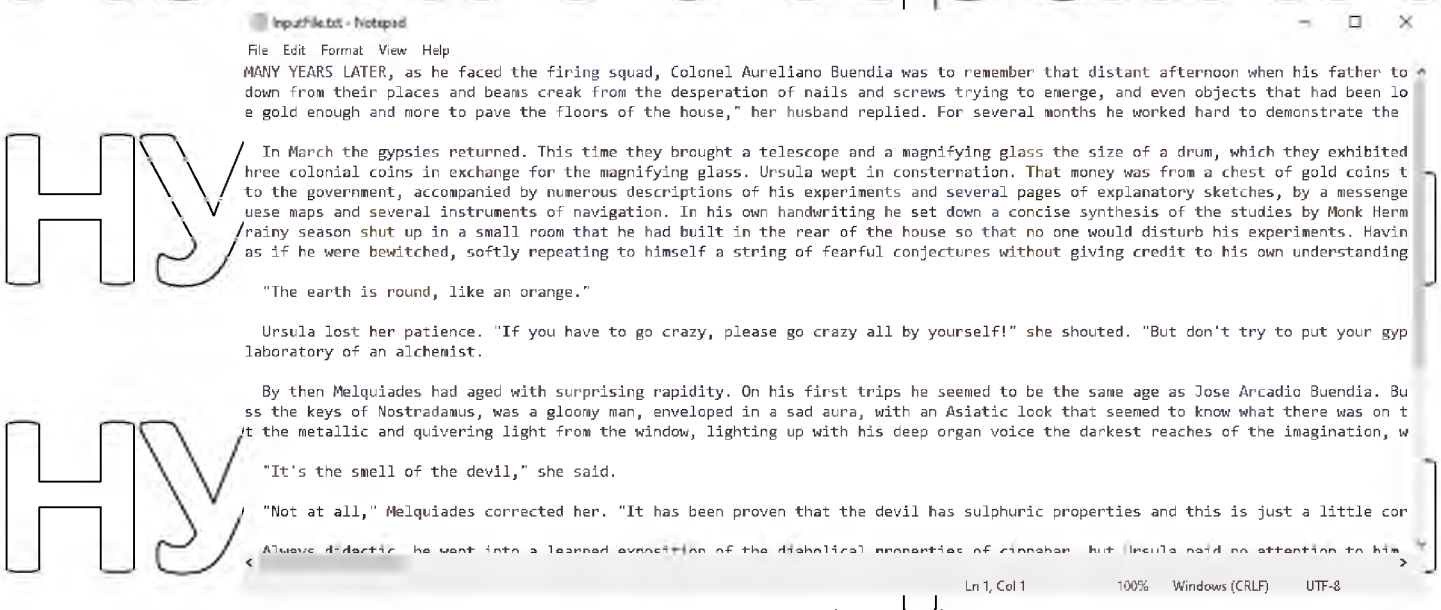

Рис.3.17. Вхідний файл

Ключ (128 біт) наведений на рисунку 3.18.

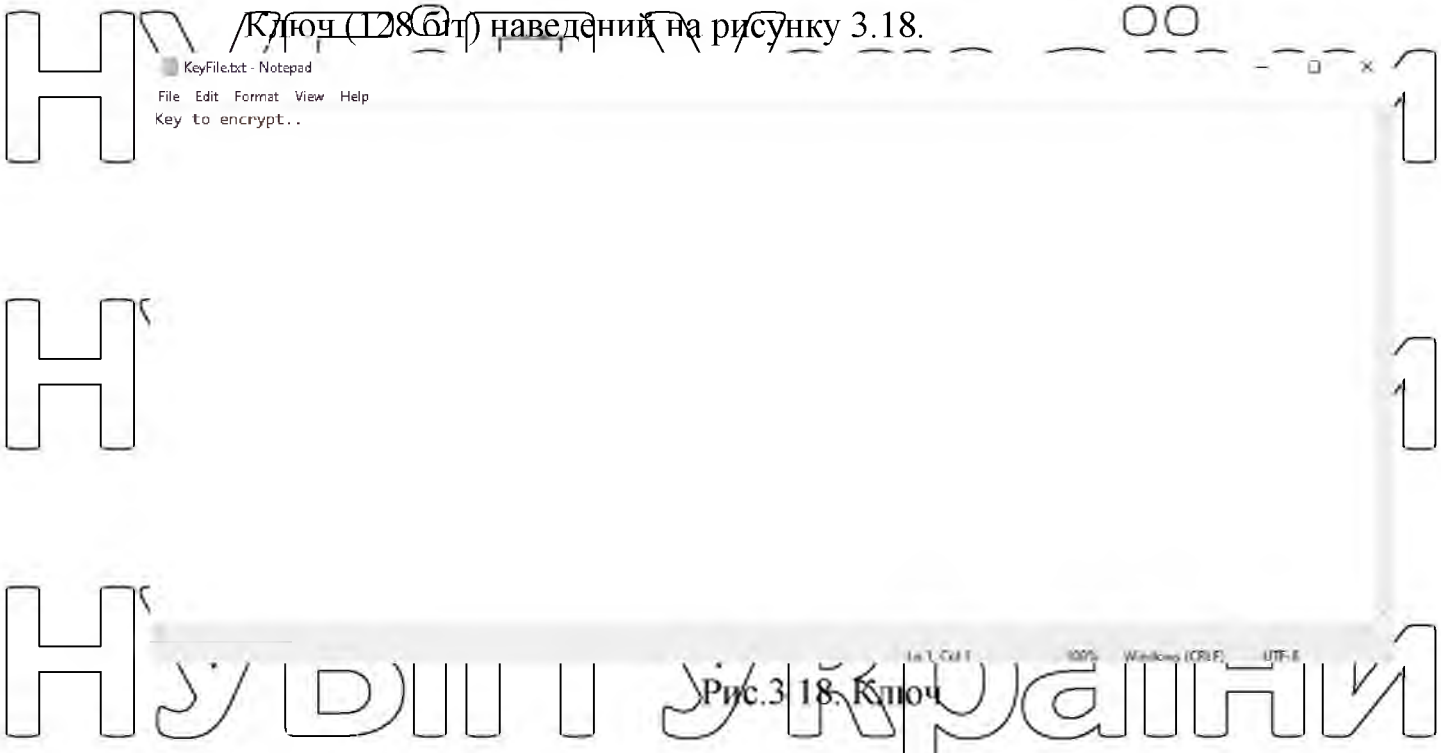

Рис.3.18. Ключ

Вихідний файл після шифрування представлений на рис.3.19.

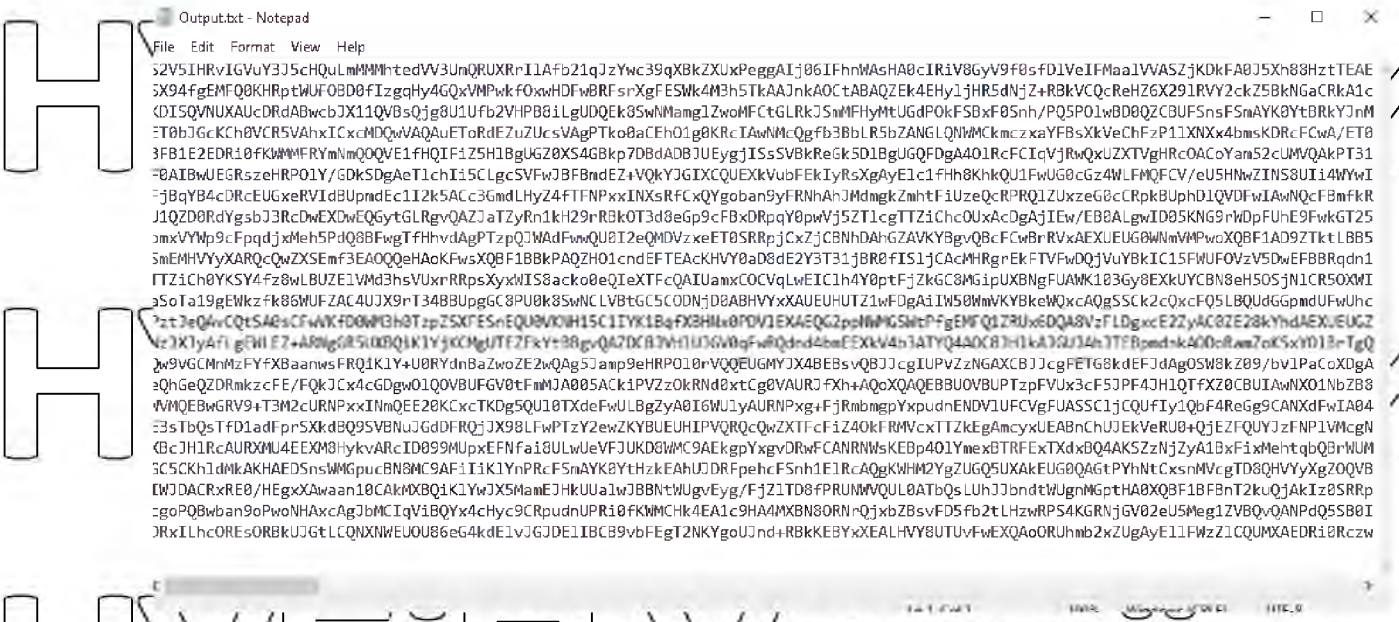

Рис.3.19 Вихідний файл після шифрування

Вихідний файл після дешифрування представлений на рис.3.20.

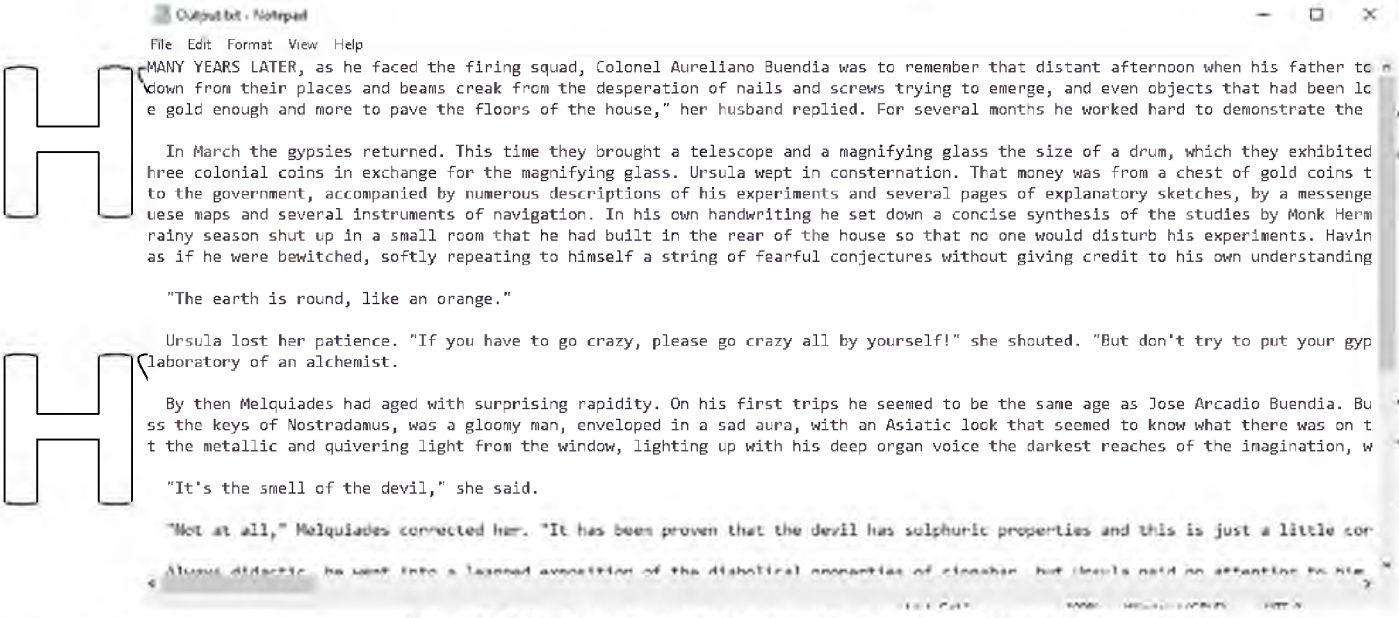

Рис.3.20. Вихідний файл після дешифрування

Приклад 3 (бінарні файли). Для перегляду бінарних файлів було використано програму Hex Editor Neo.

Вхідний файл наведений на рисунку 3.21.

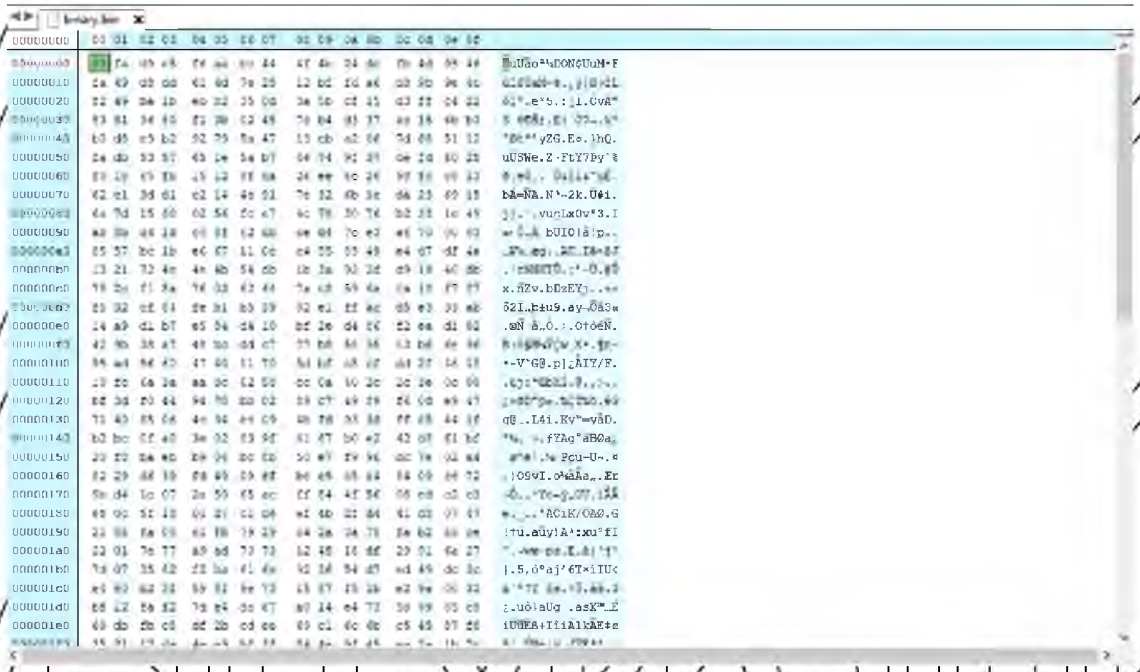

Рис.3.21. Вхідний файл

Ключ (128 біт) наведений на рисунку 3.22.

Рис.3.22. Ключ (128 біт)

Вихідний файл після шифрування наведений на рисунку 3.23.

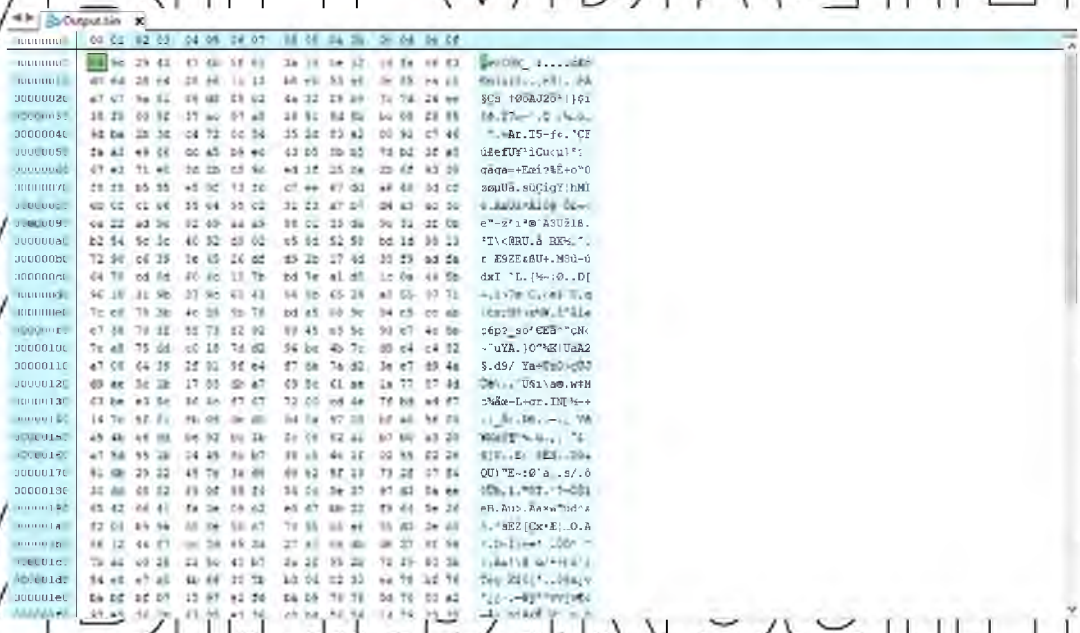

Рис.3.23. Вихідний файл після шифрування

Файл після дешифрування наведений на рис.3.24.

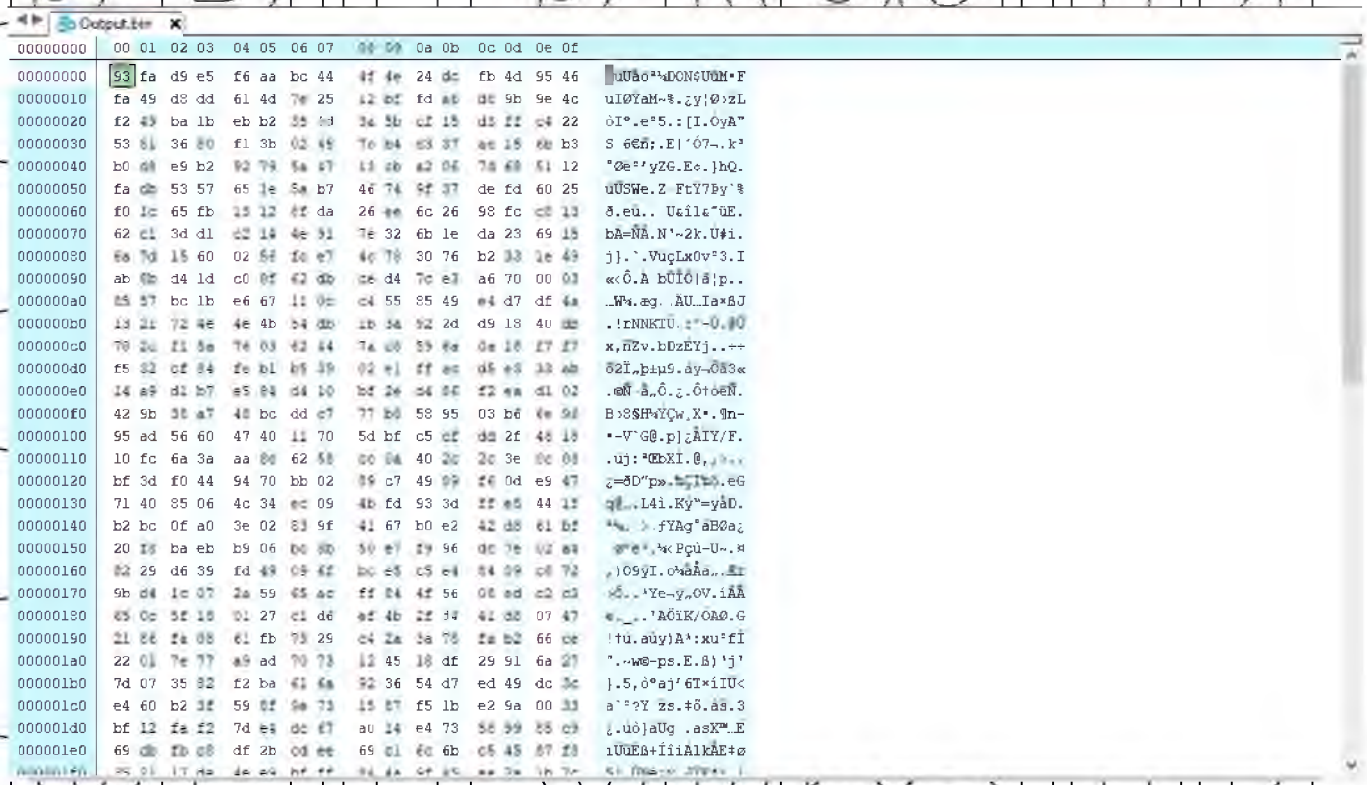

Рис.3.24. Файл після дешифрування

Представлені приклади показують працездатність розробленої системи шифрування дискретних даних.

Лістинг програми для реалізації розробленого алгоритму наведений у додатку А.

3.3 Порівняння швидкодії розробленої системи шифрування дискретних даних з відомими алгоритмами шифрування

Виконаємо порівняння швидкодії розробленої системи шифрування дискретних даних з відомими алгоритмами шифрування.

Залежність часу шифрування від обсягу даних наведена на рисунку 3.25.

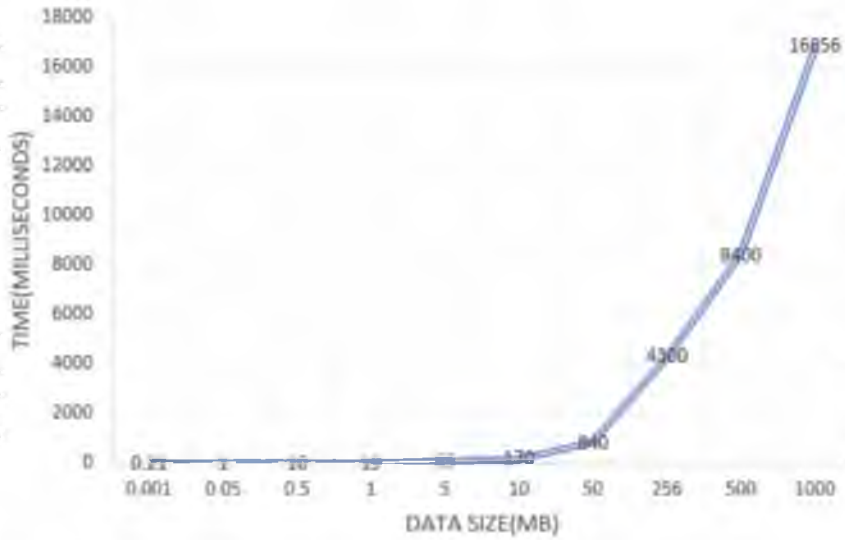

Рис 3.25. Залежність часу шифрування від обсягу даних

Час шифрування зростає при зростанні обсягу даних, що є цілком прогнозованим.

Порівняння швидкодії наведено у вигляді таблиці 3.1.

Таблиця 3.1 – Порівняння швидкодії розробленої системи з відомими алгоритмами шифрування

Algorithm	Megabytes (2^{20} bytes) Processed	Time Taken	MB/Second
Blowfish	256	3.976	64.386
Rijndael (128-bit key)	256	4.196	61.010
Rijndael (192-bit key)	256	4.817	53.145
Rijndael (256-bit key)	256	5.308	48.229
Rijndael (128) CTR	256	4.436	57.710
Rijndael (128) OFB	256	4.837	52.925
Rijndael (128) CFB	256	5.378	47.601
Rijndael (128) CBC	256	4.617	55.447
DES	128	5.998	21.340
(3DES)DES-XEX3	128	6.159	20.783
(3DES)DES-EDE3	64	6.499	9.848
Custom (128-bit key)	256	4.285	59.743

ВИСНОВКИ

В результаті написання магістерської дисертації проведено розробку інтелектуальної системи швидкого шифрування дискретних даних.

Розглянуто інтерфейс системи, а також проведено тестування розробленої системи. Приклади шифрування та дешифрування дискретних даних показують працездатність розробленого алгоритму.

Перевагами розробленої системи є наступні:

- надійність системи. Дана система використовує ефективний та швидкодіючий алгоритм шифрування;
- зручність системи. Дана система розроблена з простим та зручним інтерфейсом.

В результаті порівняння швидкодії розробленої систем з існуючими алгоритмами шифрування дискретних даних встановлено, що розроблений алгоритм є достатньо швидким та може конкурувати з іншими існуючими аналогами.

Таким чином, розроблену системи шифрування дискретних даних можна застосовувати для організацій, які потребують швидкого шифрування.

В результаті написання магістерської дисертації були виконані наступні задачі:

- виконано огляд літератури з теми дослідження. Наведено існуючі алгоритми шифрування дискретних даних;
- наведене моделювання інтелектуальної системи шифрування дискретних даних;
- представлено результати дослідження інтелектуальної системи шифрування дискретних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Франчук В.М. Захист інформаційних ресурсів: криптографічні та стеганографічні методи захисту даних. Посібник для викладачів, вчителів та студентів інформатичних спеціальностей. – К.: НПУ імені М.П. Драгоманова, 2012. – 120 с.

2. Бабаш, А.В. Криптографические методы защиты информ.: Учебное пособие. Т.1 / А.В. Бабаш. - М.: Риор, 2018. - 48 с.

3. Запечников, С.В. Криптографические методы защиты информации: Учебное пособие / С.В. Запечников, О.В. Казарин, А.А. Тарасов. - Люберцы: Юрайт, 2016. - 309 с.

4. Технології захисту інформації [Електронний ресурс] : підручник для студ. спеціальності 122 «Комп'ютерні науки», спеціалізацій «Інформаційні технології моніторингу довкілля», «Геометричне моделювання в інформаційних системах» / Ю. А. Тарнавський; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,04 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 162 с.

5. Рябко, Б.Я. Криптографические методы защиты информации / Б.Я. Рябко, А.Н. Фионов. - М.: ГЛТ, 2013. - 229 с.

6. Остапов С. Е. О-76 Технології захисту інформації : навчальний посібник / С. Е. Остапов, С. П. Євсєєв, О. Г. Король. – Х. : Вид. ХНЕУ, 2013. – 476 с.

7. Васильева, И.Н. Криптографические методы защиты информации: Учебник и практикум для академического бакалавриата / И.Н. Васильева. - Люберцы: Юрайт, 2016. - 349 с.

8. Бабаш, А.В. Криптографические методы защиты информации.Т. 1. Криптографические методы защиты информации: Учебно-методическое пособие / А.В. Бабаш. - М.: ИЦ РИОР, НИЦ Инфра-М, 2013. - 413 с.

9. Пономаренко Н.Н. Защита информации в телекоммуникационных системах: учеб. пособ. / Н. Н. Пономаренко. – Х.: Нац. аэрокосм. ун-т им. Н.Е. Жуковского «Харьк.авиац.ин-т», 2015. – 40 с.

10. How does RSA work? [Електронний ресурс] : Режим доступу: https://medium.com/hackernoon/how-does-rsa-work-f44918df914b (дата звернення: 22.11.2021).

11. Алгоритм Диффи-Хеллмана [Електронний ресурс] : Режим доступу: http://kaf403.rloc.ru/POVS/Crypto/DiffieHellman.html (дата звернення: 22.11.2021).

12. Какие алгоритмы шифрования наиболее безопасны? [Електронний ресурс] : Режим доступу: https://bezopasnik.info (дата звернення: 23.11.2021).

13. Сохацький Б.Д., Лялецький О.В. Про надійність та ефективність криптографічних алгоритмів. Інтернет-конференції НУБіП України «Теоретичні та прикладні аспекти розробки комп'ютерних систем» 2021. – 3 с.

14. Шифрование данных: основные требования к системе (на примере продукта Chameleon) [Електронний ресурс] : Режим доступу: https://shindler.ru/about/delimsya-opytom/05 (дата звернення: 23.11.2021).

15. Visual Studio IDE Review [Електронний ресурс] : Режим доступу: https://reviews.financesonline.com/p/visual-studio-ide/ (дата звернення: 23.11.2021).

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ РЕАЛІЗАЦІЇ АЛГОРИТМУ ШИФРУВАННЯ ДИСКРЕТНИХ ДАНИХ

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;
namespace kripto_vert
{
    public class MainFunctionality
    {
        private int Rest(int i, int j)
        {
            return Math.Abs(i) % Math.Abs(j);
        }
        public byte[] Encrypt(byte[] A, byte[] S, out
string time, string output = null)
        {
            int q = S.Length - 1;
            int n = A.Length;
            byte[] C = new byte[n + q + 1];
            int r;
            int m;
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            Array.Copy(S, 0, C, 0, S.Length);
            Array.Copy(A, 0, C, q + 1, A.Length);
            r = q + 1;
            m = Rest(C[r - 1], r);
            C[r] = (byte)(C[r] ^ C[m]);
            for (++r; r < q + n + 1; r++)
            {
                m = Rest(C[r - 1], r);
                C[r] = (byte)(A[r - q - 1] ^ C[r - 1] ^
C[m]);
            }
            stopwatch.Stop();
            time = stopwatch.Elapsed.TotalMilliseconds
+ " ms";
            var t = Encoding.UTF8.GetString(C);
            if (output != null)
            {
                if (Path.GetExtension(output) !=
".txt")
```

```
            {
                string res =
Convert.ToBase64String(C);
                using (StreamWriter file = new
StreamWriter(output, append: true))
                {
                    file.WriteLine(res);
                }
            }
            else
            {
                using (BinaryWriter file = new
BinaryWriter(File.Open(output, FileMode.OpenOrCreate)))
                {
                    file.Write(C);
                }
            }
            return C;
        }
        public byte[] Decrypt(byte[] C, byte[] S, out
string time, string output = null)
        {
            int q = S.Length - 1;
            int n = C.Length - q + 1;
            byte[] A = new byte[n + q + 1];
            int r;
            int m;
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            r = q + 1;
            m = Rest(C[r - 1], r);
            A[r - q - 1] = (byte)(C[r] ^ C[m]);
            for (++r; r < C.Length; r++)
            {
                m = Rest(C[r - 1], r);
                A[r - q - 1] = (byte)(C[r] ^ C[r - 1] ^
C[m]);
            }
            stopwatch.Stop();
            time = stopwatch.Elapsed.TotalMilliseconds
+ " ms";
            if (output != null)
            {
```

```
            if (Path.GetExtension(output) ==
".txt")
            {
                string res =
Encoding.UTF8.GetString(A).Replace("\0", "");
                using (StreamWriter file = new
StreamWriter(output, append: true))
                {
                    file.WriteLine(res);
                }
            }
            else
            {
                using (BinaryWriter file = new
BinaryWriter(File.Open(output, FileMode.OpenOrCreate)))
                {
                    file.Write(C);
                }
            }
        }
        return A;
    }
}
```